НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Приладобудівний факультет
Кафедра інформаційно-вимірювальних технологій

«На правах рукопису»
УДК 681.518.3

«До захисту допущено»
Завідувач кафедри
_____ Володимир ЄРЕМЕНКО
(підпис)
“___”_____2020р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 152 «Метрологія та інформаційно-вимірювальна техніка»
на тему: «Система контролю якості повітря для енергоефективної вентиляції інтелектуальних будівель»
Виконав: студент VI курсу, групи ПА–91мп
(шифр групи)

Семенко Дмитро Миколайович _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник
доцент кафедри ІВТ, к.т.н., Стаценко О. В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант_____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент_____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет Приладобудівний факультет

(повна назва)

Кафедра Інформаційно-вимірювальних технологій

(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою «Інформаційні вимірювальні технології та системи»

Спеціальність 152 «Метрологія та інформаційно-вимірювальна техніка»

(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Володимир ЄРЕМЕНКО
(підпис)

«___»_____20__ р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Семенку Дмитру Миколайовичу

(прізвище, ім’я, по батькові)

1. Тема дисертації: Система контролю якості повітря для енергоефективної вентиляції інтелектуальних будівель

Науковий керівник дисертації Стаценко Олексій Володимирович, к.т.н., доц

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Термін подання студентом дисертації: 09.12.2020 р.

3. Об’єкт дослідження: Процес контролю параметрів якості повітря в приміщення інтелектуальних будівель для забезпечення високої енергетичної ефективності роботи вентиляційних систем.

4. Предмет дослідження: Система контролю параметрів якості повітря в приміщенні та підсистема керування вентиляцією та диспетчеризації інтелектуальної будівлі.

5. Перелік питань, які потрібно розробити:

1. Аналіз існуючих підходів енергоефективного керування вентиляцією будівель. 2. Розробка структури системи контролю параметрів якості повітря в приміщені та підсистем керування вентиляцією та диспетчеризації. 3. Розробка програмного забезпечення для функціонування розроблених систем. 4. Розробка стартап проекту.

6. Орієнтовний перелік ілюстративного матеріалу:

Презентація.

7. Орієнтовний перелік публікацій:

2 наукові статті за темою магістерської дисертації.

8. Консультанти розділів дисертації*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Стартап-проект	Бояринова К.О., к.е.н., доцент кафедри менеджменту		

9. Дата видачі завдання 01.09.2020

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Огляд існуючих рішень для систем контролю якості повітря	до 02.11.2020 р.	
2.	Розробка структури пристрою управління вентиляцією	до 10.11.2020 р.	
3.	Розробка програмного компоненту	до 18.11.2020 р.	
4.	Розробка стартап-проекту	до 26.11.2020 р.	
5.	Оформлення пояснювальної записки та ілюстративного матеріалу	до 09.12.2020 р.	

Студент _____ Дмитро СЕМЕНКО
(підпис) (ім'я, прізвище)

Науковий керівник дисертації _____ Олексій СТАЦЕНКО
(підпис) (ім'я, прізвище)

РЕФЕРАТ

Магістерська дисертація на тему: «Система контролю якості повітря для енергоефективної вентиляції інтелектуальних будівель», 105 сторінок, 2 додатка, 20 джерел.

Об’єкт дослідження: Процес контролю параметрів якості повітря в приміщенняx інтелектуальних будівель для забезпечення високої і енергетичної ефективності роботи вентиляційних систем.

Предмет дослідження: Система контролю параметрів якості повітря в приміщенні та підсистема керування вентиляцією та диспетчеризації інтелектуальної будівлі.

Мета роботи: Аналіз та розробка рішення системи контролю якості повітря для енергоефективної вентиляції.

Методи дослідження та апаратура: Робота з технічною документацією, аналіз. Датчик вуглекислого газу MH-Z19B, одноплатний комп’ютер Raspberry Pi Zero W, ноутбук, фреймворк ASP.NET Core, база даних PostgreSQL, програмний засіб для організації передачі даних у реальному часі SignalR, фреймворк Blazor.

Результати роботи та сучасність продукту: Розроблено систему контролю якості повітря для керування вентиляцією. Розроблено програмний компонент для зчитування та відправки даних на базі комп’ютера Raspberry Pi Zero W та датчику MH-Z19B. Використовуючи фреймворк ASP.NET Core, бібліотеки SignalR та реляційну бази даних PostgreSQL розроблено серверну частину програмного забезпечення для зберігання та обробки даних.

Рекомендації щодо використання результатів роботи: Результати розробки можуть бути використані для створення енергоефективної вентиляції.

ВЕНТИЛЯЦІЙНА СИСТЕМА, ВИМІРЮВАННЯ, ЯКІСТЬ ПОВІТРЯ, ДАНІ В РЕЖИМІ РЕАЛЬНОГО ЧАСУ, MH-Z19B

ABSTRACT

Master's thesis: "Air quality control system for energy efficient ventilation of intelligent buildings", 105 pages, 2 applications, 20 sources

Object of research: The process of controlling the parameters of air quality in the premises of intelligent buildings to ensure high energy efficiency of ventilation systems.

Subject of research: Indoor air quality control system and ventilation control and dispatching subsystem of intelligent building.

Objective: Analysis and development of air quality control system solution for energy efficient ventilation.

Research methods and equipment: Work with technical documentation, analysis. MH-Z19B carbon dioxide sensor, Raspberry Pi Zero W single-board computer, laptop, ASP.NET Core framework, PostgreSQL database, software for real-time data transmission SignalR, Blazor framework.

Results of work and their novelty: An air quality control system for ventilation control has been developed. A software component for reading and sending data based on the Raspberry Pi Zero W computer and the MH-Z19B sensor has been developed. Using the ASP.NET Core framework, the SignalR library and the relational database PostgreSQL, the server part of the software for data storage and processing was developed.

Recommendations on the use of work results: The results of the development can be used to create energy-efficient ventilation.

VENTILATION SYSTEM, MEASUREMENTS, AIR QUALITY, REAL-TIME DATA, MH-Z19B

ЗМІСТ

ВСТУП

Вентиляція - головний елемент в створенні сприятливого клімату, покликаний для подачі свіжого повітря з вулиці і видалення забрудненого повітря з приміщень. Повітря в приміщеннях - важливий фактор, що впливає на здоров'я, і, як наслідок, на працездатність людей, в які перебувають цих приміщеннях. Також набирають популярність та поширеність інтелектуальні будівлі, які характеризуються високим рівнем автоматизації, що дозволяє суттєво зменшити витрати енергії та ресурсів для їх обслуговування. Неабияку роль в цьому відіграє використання енергозберігаючих технологій.

Правильна експлуатація вентиляційних систем дозволяє зберігати, як теплову так і електричну енергію. Для цього використовуються різні рішення, але ключовим в цьому процесі є регулювання продуктивності вентиляції по потребі. Це дозволяє забезпечити якість повітря в приміщеннях відповідно до нормативів, здійснюючи одночасно економію енергії. Для побудови такої системи необхідно вирішити значну кількість задач. Однією з основних з цих задач є побудова системи контролю якості повітря в приміщеннях будівлі, яка б забезпечувала інтеграцію в загальну систему диспетчеризації будівлі, була масштабованою, легко налаштованою, дозволяла б не лише відображати поточні дані, але б надавала змогу зберігати їх та оброблювати.

Незважаючи на те, що подібні системи існують, їх виробники не розкривають принципи їх побудови та функціонування. Тому вирішення задач, пов’язаних з розробкою структури таких систем та їх програмним забезпеченням є актуальними.

Для вирішення цих задач необхідно провести аналіз особливостей побудови систем вентиляції для інтелектуальних будівель, визначити структуру системи контролю якості повітря, розробити програмне забезпечення роботи такої системи.

1 СТРУКТУРА СИСТЕМИ ВЕНТИЛЯЦІЇ В ІНТЕЛЕКТУАЛЬНИХ БУДІВЛЯХ

1.1 Інтелектуальні будівлі

Інтелектуальне проектування будівель - це майбутнє будівельних галузей. Більшість сучасних громадських та житлових будівель планується з метою зменшення витрат за рахунок зменшення споживання енергії. Посилення стратегій енергозбереження та використання підходів до сталого проектування є необхідними факторами розвитку цієї галузі. В даний час намітився поступовий перехід до зведення так званих «розумних», або інтелектуальних, будівель для підвищення якості середовища, економії матеріалів і енергії. Елементи «розумних» систем будівель відомі вже давно, але системи забезпечення комфортних умов для жителів і швидкого реагування на зміну потреб за допомогою вбудованих керуючих пристроїв з'явилися недавно. Поняття інтелектуальна будівля з'явилося в США в 1970 80х роках. Елементи інтелектуальності сьогодні притаманні майже будь-якій будові. Але все-таки інтелектуальний будинок - поняття зовсім іншого масштабу. Це будівля, яка може керувати системи, щоб забезпечити мешканцям оптимальні сервіси надійним, економним і раціональним способом. Ця будівля, в якому встановлені відповідні обладнання та системи, що дозволяють виконувати моніторинг, прогноз і управління. Вони приєднані до «розумних» систем електропостачання, які враховують використовувану в даний час тарифну схему і мінімізують споживання в години пік. Вони здатні покривати хоча б частково свої потреби в електроенергії, збирають дощову воду і мінімізують витрати води і, нарешті, що найбільш важливо, вони мають інструменти з призначеним для користувача інтерфейсом, щоб надавати менеджерам і навіть жителям параметри будівлі в режимі реального часу і можливість впливати на них [1].

Зазвичай розуміють інтеграцію в єдину систему управління будівлею наступних систем:

- систем електропостачання, опалення, вентиляції та кондиціонування;
- служб безпеки (протипожежної, антисейсмічного, охорони будинку, охоронно-пожежну сигналізацію, систему контролю доступу в приміщення, контроль протікання води, витоку газу і т. д.);
- різних телекомунікаційних мереж (мережі зв'язку, в тому числі супутникової, оптико-волоконні кабельні мережі і т. д.);
- автоматизації внутрішнього транспорту (наприклад, ліфтів);
- централізованого збору та утилізації відходів;
- автоматизації системи контролю якості внутрішнього середовища будинку і деякого обсягу зовнішнього простору;
- телеметрія - віддалене спостереження за системами;
- механізацію будівлі (відкриття / закриття воріт, шлагбаумів, і т. д.);

Створення оптимального середовища, забезпечення комфортних умов діяльності, зниження витрат на експлуатацію - це основні критерії концепції інтелектуальної будівлі. «Розумна» будівля через систему датчиків контролює стан зовнішнього і внутрішнього середовища і при відхиленні показників від норми включає пристрої, які очищають, наприклад, середу від забруднень або поліпшують інші показники. Одним з основних компонентів інтелектуальної будівлі є система автоматизованого управління експлуатацією будівлі, основним завданням якого є забезпечення надійного і гарантованого керування всіма системами, що знаходяться в експлуатації будівлі, і виконавчими пристроями. Система здатна за рахунок повної інформації від всіх експлуатованих підсистем, будь то пожежна охорона, система телеспостереження, телефонія, водопостачання, електроживлення, кондиціонування, прийняти правильне рішення і виконати відповідну дію, поінформувати відповідну службу про подію. Система повинна вміти

розпізнавати певні ситуації і яким-небудь чином на них реагувати таким чином, щоб найбільш ефективним способом забезпечити безпечне і комфортне перебування в будівлі, зменшивши до мінімуму споживання енергії та енергоресурсів. Будівля проектують таким чином, щоб всі системи його управління могли інтегруватися один з одним з мінімальними витратами, а їх обслуговування було б організовано оптимальним чином. «Розумними» можуть бути будь-які будівлі - житлові будинки, установи, виробничі об'єкти і ін. Функції «розумних» будинків можуть бути дуже широкими - від контролю середовища і стану жителів до контролю стану самої будівлі. Новим типом «розумного» будинку широкого призначення може стати будівля з автоматичним стеженням за станом своїх конструкцій. Така будівля за допомогою систем датчиків стежить за деформацією і станом конструкцій.

1.2 Можливості енергозбереження в будівлях

Енергозбереження означає раціональне використання енергії у всіх ланках перетворення енергії - від видобутку первинних енергоресурсів до споживання всіх видів енергії кінцевими користувачами. Заходи з енергозбереження можуть бути різними. Один з найбільш дієвих способів збільшення ефективності використання енергії - застосування сучасних технологій енергозбереження. Технології енергозбереження не тільки дають значне зменшення витрат на енергетичні витрати, а й мають очевидні екологічні плюси. Є багато вимог для збільшення енергоефективності в будівлях. Наприклад ось декілька з них:

- Зниження теплових втрат в будівлях
- Оптимізація системи опалення будівель
- Використання сонячної енергії в інженерних системах будинків
- Енергозбереження та ресурсозбереження при експлуатації систем водопостачання, водовідведення

- Раціоналізація енергоспоживання при використанні електротехнічних приладів
- Використання систем управління інженерними системами будівель

1.2.1 Зниження теплових затрат

Споживання теплової енергії для опалення будівель становить значну частку в балансі енергоспоживання. З урахуванням використання теплової енергії для гарячого водопостачання, а також для адміністративних і виробничих будівель, можна оцінити частку теплової енергії, що спрямовується на опалення близькою до 55-60%. Теплоізоляція та герметизація будівель є досить привабливими напрямками в плані зниження втрат теплової енергії при опаленні будівель. Тривалий час будівництво будівель здійснювалося з мінімальним використанням теплоізоляційних елементів в них. Такий підхід гуртувався на можливості отримання відносно дешевої енергії для опалення. В даний час випускається велика кількість видів тепло ізолюючих матеріалів на основі мінеральної вати, полістиролу, пінопласту та інших мінеральних синтетичних матеріалів. Дедалі більшого поширення набувають теплоізоляційні матеріали, вироблені на основі натуральних інгредієнтів (целюлози, льону та інших). Теплоізоляція може проводитися за рахунок використання багатошарової конструкції огороджувальних стін, коли утеплювач укладається між шарами несучих конструкційних матеріалів, або шляхом кріплення утеплювача на зовнішніх стінах і (або) всередині приміщень. З більш простих способів теплової модернізації будівель в даний час використовується часткова або повна теплоізоляція зовнішніх стін, а також і зменшення тепловтрат через вікна шляхом використання віконних блоків з підвищеними теплоізоляційними властивостями.

Найбільш поширеним способом модернізації вікон є заміна традиційних конструкцій віконних прорізів на герметичні. Установка герметичного вікна

знижує втрати за рахунок зменшення припливу холодного повітря через вікно і підвищення опору передачі тепла через площу склопакета. Склопакети виготовляються з блоку, що складається з двох і більше шибок, між якими встановлена дистанційна рамка. По всьому периметру склопакета по краях монтується спеціальний профіль, який склеюється з шибками через двоступеневе ущільнення для пароізоляції та забезпечення герметичності конструкції. У простір між шибками не повинно потрапляти повітря, оскільки це призводить до запотіванню вікна і втрати прозорості. Для посилення ефекту теплоізоляції простір між шибками може заповнюватися інертним газом - аргоном. Існують конструкції склопакетів, у яких в просторі між шибками створюється вакуум, а на внутрішню сторону скла наноситься невидимий шар срібла, що зберігає тепло.

1.2.2 Оптимізація системи опалення в будівлях

Конструктивно системи водяного опалення можуть виконуватися по різноманітним схемам. Так, подають магістральні трубопроводи для розподілу нагрітої води можуть пролягати у верхній або в нижній частині будівлі, виконуватися з попутним рухом води або тупиковими. Від магістральних трубопроводів вода може подаватися до нагрівальних приладів по одній або двом трубам (так звані однотрубні і двотрубні системи). Підвідні трубопроводи до нагрівальних приладів можуть розташовуватися горизонтально або вертикально. Ці опалювальні системи є більш гнучкими і створюють умови для енергозбереження. Також для підвищення енергоефективності вдало підходить встановлення режиму роботи термостатичного регулятора виробляється ручним перемикачем. При підвищенні температури повітря в опалювальному приміщенні збільшується обсяг чутливого до температури рідинного елемента, який переміщує шток, що впливає на замикає клапан. Клапан зменшує прохідний перетин трубопроводу для подачі теплоносія і знижує його витрату через прилад.

Наступним кроком в напрямку модернізації терморегуляторів є використання електронних конструкцій. Електронні терморегулятори забезпечують реалізацію більш складних програм регулювання, наприклад, програмування зміни температури в опалювальному приміщенні за часом - установка зниженої температури в період відсутності людей в приміщенні або в нічний час, підвищення температури в приміщенні на час приїзду людей і т. д.

1.2.3 Використання систем управління інженерними системами будівель

Ще 10-15 років тому для контролю та управління інженерними системами вважалося достатнім наявність найпростіших контрольно-вимірювальних приладів, а також механізмів регулювання. В міру ускладнення систем життєзабезпечення будівель удосконалювалися і пристрої для їх управління, наприклад, сьогодні управління системою опалення враховує температуру зовнішнього повітря, наявність людей в приміщенні і ряд інших параметрів. Сучасні локальні системи управління повинні включати ряд датчиків, пристроїв захисту і інших механізмів.

З огляду на взаємозв'язок між окремими системами життєзабезпечення будівель, до кінця 20 століття будівельна наука поповнилася такими новими поняттями як «Smart home», «Intelligent Building» («розумний дім»), які позначають новий напрям в архітектурі та інженерії. Основні принципи таких будівель засновані на раціональному витрачанні енергоресурсів, об'єднанні інженерного обладнання та систем життєзабезпечення в єдиний керований комплекс, наявність програмно-апаратних засобів автоматичного та дистанційного керування (так зване інтелектуальне управління). Головною особливістю такого типу управління є об'єднання окремих систем в єдиний керований комплекс. При цьому за рахунок інтеграції інформації, що надходить від усіх експлуатованих підсистем (клімат-контролю, протипожежного захисту, відеоспостереження, системи водопостачання та водовідведення, електропостачання, освітлення, зв'язку та інших), швидко

приймаються обґрунтовані рішення і виконуються необхідні дії, пов'язані з експлуатацією будівлі. Будівля має проектуватися таким чином, щоб всі системи управління могли інтегруватися один з одним з мінімальними витратами і з забезпеченням можливості нарощування і видозміни конфігурації змонтованих систем. Управління може здійснюватися диспетчерським центром, розміщеним в обслуговуваному приміщенні або групою будинків, а також віддалено шляхом використання спеціальних ліній зв'язку або інтернету. Дуже перспективні розробки в частині використання інтернет технологій для контролю і управління. Інтернет увійшов в наше життя як глобальна інформаційне середовище, що відкриває принципово нові можливості. Однак до сих пір інтернет-технології не знаходили широкого застосування в промисловості. З появою сигнальних модемів з'явилася можливість управляти різними приводами дистанційно, навіть на великих відстанях. Завдяки цим пристроям організувати доступ до управління тепер можна з будь-якого комп'ютера, підключеного до мережі, або з мобільного телефону. При цьому гарантується така ж безпеку віддаленого доступу, як і при електронних операціях з банківським рахунком.

Для підвищення ефективності енергозберігаючих заходів велике значення має не тільки впровадження нового обладнання, передових технологій, але і чітко організоване управління енергоспоживанням, тобто енергоменеджмент і енергоаудит на підприємствах, в організаціях та будівлях. Енергетичний менеджмент являє собою сукупність технічних і організаційних заходів, спрямованих на підвищення ефективності використання енергоресурсів і є частиною загальної структури управління підприємством. Основне завдання його полягає в проведенні комплексного аналізу енергоспоживання і на його основі - проведення енергозберігаючих заходів на підприємстві [2].

1.3 Забруднення повітря в будівлях, вимоги до вентиляції, можливості енергозбереження за допомогою вентиляції

Свіже, безпечне для людей повітря - базовий фактор, що визначає загальне самопочуття і стан здоров'я. У виробничих і складських комплексах до якості аерації теж не можна ставитися поверхово: вона дозволяє підтримувати потрібний для обладнання або товарів режим температур і вологості, відводить з приміщень їдкі випари, пил. Тому контролюючі інстанції строго стежать, щоб при будівництві, використанні будівель будь-якого призначення дотримувалися базові вимоги до систем вентиляції, а також конкретні положення для об'єкта [3].

Комплекси повітрообміну поділяють в цілому на комфортні і технологічні. Перші повинні відповідати санітарно-гігієнічної складової вимог. Другі, крім цього, перевіряються за умовами, визначеними технологією промислового об'єкта.

Для окремих елементів вентиляції формуються конкретні положення. Основні положення:

- для витяжних систем - підтримка допустимого рівня концентрації шкідливих речовин в повітрі, що надходить в приміщення і видалення забруднених мас;
- для місцевих джерел припливної вентиляції - ефективне уловлювання з подальшим видаленням шкідливих виробничого (газ, їдкі пари і так далі) і побутового характеру.

Офісні будівлі щодня зайняті мільйонами робітників. Утримання цих будівель у сприятливому для працівників стані є головною проблемою для керівників будівель, дизайнерів та компаній, які ними володіють. Правильні вимоги до вентиляції в офісі - ключова частина вирішення проблеми. Однак різні стандарти будівельних норм та норм безпеки можуть ускладнити задання мінімальних вимог. Розглянемо деякі основні вимоги щодо вентиляції офісних будівель.

Вимоги до вентиляції офісних приміщень потребують більш складної системи, яка залежить від поєднання процесів. Є чотири основних етапи:

- Забір повітря відбувається завдяки додаванню в систему чистого повітря.
- Зовнішнє повітря проходить кондиціонування і змішується з поточним припливом повітря.
- Повітря після кондиціонування розподіляється по всій будівлі.
- Система викидає назовні або циркулює старе повітря.

Ці кроки забезпечують механічну вентиляцію. Якщо проблеми з проектуванням або обслуговуванням будівель спричиняють невиконання одного з етапів, загальна якість повітря в будівлі погіршується. Якщо якість повітря погіршується, це може спричинити загрозу безпеці праці та здоров'ю. Вимоги вентиляції офісів повинні вирішувати такі питання:

- Герметичність
- Прилади опалення
- Витоки та забруднення каналів
- Розподіл вентиляції
- Механізми контролю
- Фільтрація

Для утримання на високому рівні вентиляції офісів та забезпечення чистішого, здоровішого офісу треба дотримуватись технічних вимог перелічених вище. І якщо настав час оновлення, ці вдосконалення зрештою допоможуть збільшити загальну продуктивність вентиляції будівлі.

Є також і можливість енергозбереження у вентиляційних системах. Два найпопулярніших підходу це:

- Рекуперація тепла
- Вентиляція по потребі

Так як будь-який вентилятор має свої характеристики напору від подачі. Ця характеристика вказується виробником для кожного конкретного вентилятора і має вигляд приблизно як на наведеному рисунку.

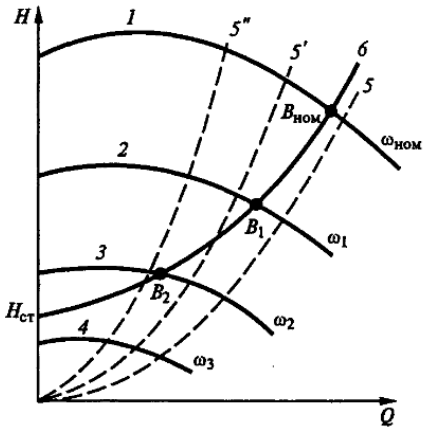

Рисунок 1.1 – Залежність тиску від подачі

На рисунку показані характеристики нагнітача при різних значеннях частоти обертання (криві 1, 2, 3, 4). Кількість поданого повітря прямо пропорційна швидкості обертання вентилятора.

$$Q \sim \omega$$

Q – подача повітря. ω - швидкість обертання вентилятора.

При цьому потужність пропорційна кубу швидкості обертання.

$$N \sim \omega^3$$

N – потужність.

Це обумовлює можливість енергозбереження при зміні кількості повітря за рахунок зміни швидкості. Якщо не змінюються параметри системи вентиляції, то ККД вентилятора залишається сталим у цьому випадку(пунктирні лінії на рисунку).

Виходячи з цього можна зробити висновок, що при зменшенні обсягу повітря, що подається до приміщення за рахунок роботи двигуна, то вдається суттєво знизити споживану двигуном енергію. Так при зменшенні подачі вентиляцією повітря в 3 рази зменшення механічної потужності в 30 разів. Не

зважаючи на те, що ККД двигуни при цьому зменшується, абсолютне значення сумарних втрат зменшується в 20 разів. Також використання підходу при якому забезпечується мінімізація струму статора є кращою аніж забезпечення постійного рівня намагнічування електричного двигуна [4].

1.4 Аналіз існуючих систем вентиляції

Системи класифікують за різними критеріями:

- способу подачі
- призначенням
- способу повітрообміну
- конструктивним виконанням

Тип вентиляції визначають на етапі проектування будівлі. При цьому беруть до уваги як економічну так, і технічну сторону а також санітарно-гігієнічні умови.

Якщо базуватися на способах подачі і вилучення повітря з приміщення можна виділити 3 категорії вентиляції:

- природна
- механічна
- змішана

Виконують проектування вентиляції, якщо таке рішення здатне забезпечити повітрообмін, відповідний встановленим нормам. Коли вентиляція природного типу не задовольняє вимоги санітарно-гігієнічних нормативів, вибирають другий варіант - механічний спосіб активації повітряної маси.

Якщо можливо в добавок до другого варіанту вентиляції частково використовувати перший, в проект закладають змішану вентиляцію. В житлових будинках приплив повітря відбувається через вікна, а витяжний

обладнання розташовують на кухні і в санітарній кімнаті. Тому важливо налагодити між приміщеннями хороший повітрообмін. Вентиляція змішаного типу. Застосовують її, коли природна вентиляція не може бути єдиним варіантом. Для якісного повітрообміну в приміщеннях з дуже забрудненим повітрям влаштовують механічну вентиляцію.

За способом повітрообміну виділяють системи вентиляції загальні і місцеві. Перша повинна забезпечувати весь обсяг приміщення достатнім повітрообміном з підтриманням всіх необхідних параметрів повітря. Додатково вона повинна видаляти надлишок вологи, тепла, забруднень. Обмін повітря може здійснюватися як по канальної, так і безканальної системі. Загально обмінна припливна вентиляція знижує рівень концентрації шкідливих речовин, що залишилися після роботи місцевої та загальної витяжної системи вентиляції. Призначення місцевої вентиляції - постачання чистим повітрям конкретних місць і видалення забрудненого з тих точок, де він утворюється. Як правило, її влаштовують у великих приміщеннях з обмеженим числом працюючих. Повітрообмін відбувається тільки на робочих місцях. Виходячи з цієї ознаки вентиляційні системи ділять на канальні і безканальні. Системи канального типу складаються з повітропроводів за якими транспортується повітря. Установка такої системи доцільна у великих за обсягом приміщеннях. Коли канали відсутні, систему називають безканальною Прикладом такої системи є звичайний вентилятор. Існує 2 види безканальних систем - стельові і ті, що прокладаються під підлогою. Безканальні системи більш прості у виконанні і споживають менше енергії.

Рух повітряних мас при природній вентиляції відбувається природним шляхом за рахунок:

- температурного перепаду всередині і зовні будівлі
- різниці тиску між приміщенням і витяжкою, розміщеної на покрівлі будівлі

- під впливом вітру

Вентиляція може бути організованою і неорганізованою. Регульована або організована система функціонує завдяки аерації або присутності дефлекторів. Аерація - це процес, під час якого повітря надходить і йде через відкриті кватирки, ліхтарі, фрамуги.

Інфільтрація або нерегульована вентиляція природна вентиляція - це потрапляння в приміщення повітря через нещільності в конструкціях.

Вентиляційна система, за допомогою якої повітря подають і видаляють з використанням додаткових процесів на значні відстані, називається механічною. Є й інші назви у цього виду вентиляції - примусова і штучна. Застосовують її як для забезпечення технологічних процесів на різних виробництвах, так і для створення комфортних умов для людини. Механічна вентиляція, на відміну від природної, не залежить від зовнішніх умов. Вона повністю підконтрольна і керована. Повітря, що подається в приміщення, проходить обробку і при налагодженій системі всі його параметри відповідають стандартам. Викиди також надходять в атмосферу вже очищеними від шкідливих включень до потрібного ступеня. Механічна система включає прилади та обладнання - вентилятори, уловлювачі пилу, нагрівачі повітря, електродвигуни. Все це споживає багато електроенергії. Наявність механічної вентиляційної системи дозволяє оптимально розподілити повітря з подачею його до конкретного місця. З її допомогою шкідливі викиди вловлюють у джерела їх утворення не дозволяючи забруднити повітря всього приміщення. Недолік механічної вентиляції - великі фінансові вкладення при її монтажі та експлуатації. Щоб користуватися всіма її перевагами, доведеться боротися із забрудненням каналів, регулярно виконувати заміну фільтрів. Якщо встановлений вентиляційний фільтр з функцією рекуперації тепла, потрібно перед настанням літнього періоду переходити на літній вкладиш. Якщо залишити

зимовий варіант, він буде знижувати ефективність вентиляції. Заміщення повітря в канальної системі відбувається за допомогою вентиляторів - відцентрових або осьових, ежекторних установок. Конструкції з механічним приводом можуть бути як припливними, так і витяжними. Припливну вентиляцію іноді виконують спільно з центральним опаленням. Приймач повітря в такій системі може мати вигляд отворів в огороджувальних конструкціях будівлі, що стоїть окремо або приставлений до шахти. При монтажі за межами будівлі шахта-повітрозабірник знаходиться над рівнем землі або на даху.

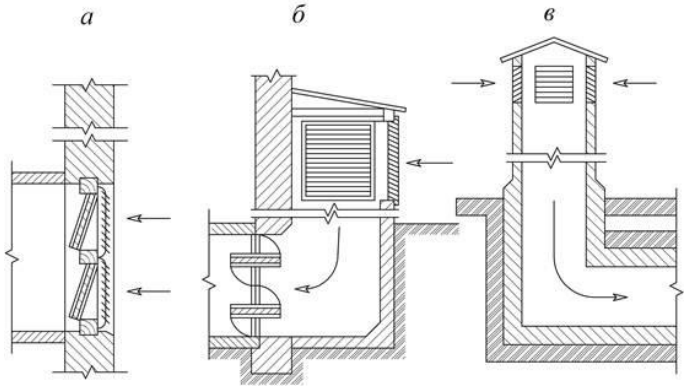

Рисунок 1.2 – Схема розташування приймачів повітря

Схема розташування приймачів повітря: в захисної конструкції (а), біля зовнішньої стіни (б), на покрівлі (в). Канали для витяжки та шахти утеплюють зовні, інакше взимку в них буде з'являтися полій. На вибір конструкції і місця знаходження приймачів повітря впливають вимоги, що пред'являються до ступеня чистоти зовнішнього повітря, а також особливості архітектури будівлі. Низ отвору, через який надходить чисте повітря, повинен розташовуватися на дистанції мінімум 2 м від землі, а в разі дислокації будівлі в зеленій зоні - 1 м. Зовнішні приймачі повітря не можна розміщувати там, де є шкідливі викиди. Повітряні маси потрапляють в шахту за допомогою вентилятора. Проходячи через калорифер, вони нагріваються, зволожуються або навпаки підсушуються і надходять всередину по воздуховодам, які мають отвори. Надходження повітря може здійснюватися і

через відгалуження, оснащені насадками, направляючими припливні повітряні маси. Обсяг повітря, що подається регулюють шибери або клапани, що знаходяться в відгалуженнях.

Видалення відпрацьованого повітря з приміщення - завдання витяжної вентиляції. Видалення використаного повітря з кімнати відбувається за рахунок зниження тиску в ній. Таким чином, створюються умови для надходження в неї повітря зовні або з суміжного приміщення.

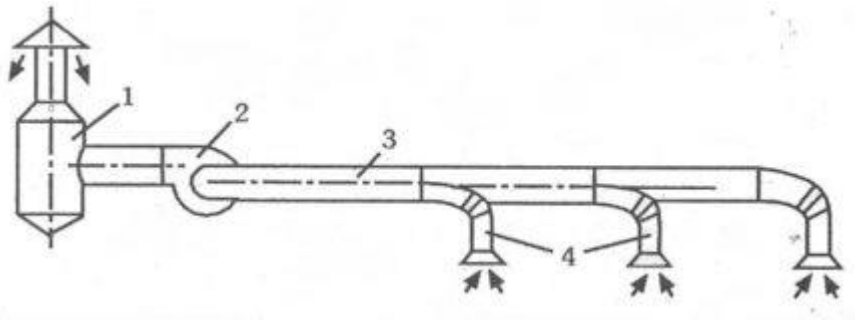

Рисунок 1.3 – Конструкція витяжної вентиляції

У конструкцію витяжної вентиляції входить очисний пристрій (1) вентилятор (2) повітроводи - центральний (3) викачуваний (4). Коли в приміщенні немає вентиляції іншого виду, крім витяжної, тиск в ньому опускається до позначки нижчою, ніж в суміжних кімнатах або нижче ніж зовні.

Базою припливно-витяжної вентиляції є 2 потока, що рухаються назустріч один одному. Вона складається з двох самостійних систем - припливної та витяжної або з одного блоку. У нього вбудовано все інше обладнання необхідне для роботи і на приплив, і на витяжку. При наявності 2 окремих систем вентиляція працює без рециркуляції і називається розімкнутої. Вентиляційну систему другого виду називають замкнутою, і працює вона з рециркуляцією. Система з рециркуляцією економить енергію, що витрачається на охолодження або нагрівання повітря, тому що повітряна маса нагрівається не повністю, а тільки той її обсяг, що надходить зовні. Повітря, що видаляється в системі з рециркуляцією повертається в

приміщення повторно з домішкою свіжого повітря, що становить 10-15% від загального обсягу повітряної маси. У регіонах з холодним кліматом замкнута система неефективна тому рециркуляційні і зовнішні повітряні маси змішуються не досить добре.

В системі вентиляції, що складається з сучасного побутового вентиляційного обладнання може бути реалізована абсолютно будь-яка схема поєднання природних і примусових пристроїв:

- повністю природна вентиляція в приміщенні - природна витяжна вентиляційна система, яку доповнюють природні припливні пристрої. Витяжки на кухні і в санвузлах, вікна та двері - найдешевша технологія вентиляційного призначення. Але стандартні вентиляційні вироби не завжди справляються зі своїм завданням, тому багато господарів приміщень вибирають такі варіанти
- природна витяжна вентиляційна система, яку доповнюють примусові припливні системи - заміщення повітря організовується за рахунок руху повітряних мас - установки припливної вентиляції вводять в приміщення повітряну масу, створюючи в приміщенні надлишковий тиск, що змушує повітря залишати приміщення
- примусове витяжний вентиляційний пристрій і стандартні вироби припливу - системи витяжної примусової вентиляції для приміщення забирають повітря, утворюючи низький тиск, що ліквідується вентиляцією системи припливу
- повністю примусова вентиляційна система - механічна витяжна вентиляція доповнюється механічної припливної. Такій вентиляційній системі забезпечувати необхідні умови в приміщенні найпростіше, проте повністю механічні вентиляційні системи і обладнання вимагають величезної кількості електроенергії

Сучасні вентиляційні системи просунулися і в цьому питанні - теплопровідні матеріали для вентиляції дозволяють проектувати і реалізовувати вентиляційну системи з рекуперацією тепла. З рекуперативної вентиляцією витрата електроенергії для власників приміщення скорочується в кілька разів. Системи з рекуперацією проектуються з теплопровідних матеріалів із загальним каналом вентиляції системи подачі та відведення. Матеріали для вентиляційної системи також вибираються ретельно при установці повітроводів вентиляції - з одних матеріалів виходять більш гнучкі повітроводи, інші матеріали дають більш міцні канали в системі [5].

1.5 Структура системи вентиляції в інтелектуальному будинку

Для забезпечення необхідних умов належного руху повітря в інтелектуальних будинках, створення надійних систем вентиляції та кондиціонування, щоб при цьому скоротити потрібність в обслуговуючому персоналі, а також для економії електроенергії і збереження холоду і тепла, вдаються до застосування автоматизованих систем кондиціонування і вентиляції, які в числі іншого дозволяють робити автоматичне відключення і включення обладнання. Щоб автоматизована система працювала правильно і найбільш економічно, для спостереження за основними параметрами на щити виносять прилади контролю. На окремих вузлах, для можливості відстеження роботи окремих елементів, встановлюють місцеві прилади контролю, для моніторингу проміжних показників.

Автоматика самописних приладів дозволяє вести облік і аналіз поточної роботи вентиляційного обладнання, а для своєчасної фіксації небезпечних відхилень служать прилади сигналізують, покликані запобігти порушенню технологічного процесу і, як наслідок, - брак продукції. Індикатори роботи системи вентиляції та кондиціонування встановлюють як в системі припливної вентиляції, так і в комбінованих системах з повітряним опаленням, і в системах кондиціонування повітря.

Однією з основних складових сучасних систем кондиціонування і вентиляції є засоби і системи автоматики. Вони реалізують різні функції управління, які повинні з одного боку забезпечити підтримання необхідного мікроклімату в приміщенні, що обслуговується, а з іншого - економічну і надійну роботу технологічного обладнання. Діапазон функцій управління, виконуваних системами автоматики за кількістю і складності реалізації досить широкий: від простого включення - виключення до централізованого управління кліматичним або всім інженерним обладнанням будівлі.

Для забезпечення нормального пуску системи кондиціонування або вентиляції слід враховувати:

- Попереднє відкриття повітряних заслінок до пуску вентиляторів. Це виконується в зв'язку з тим, що не всі заслінки в закритому стані можуть витримати перепад тисків, що створюється вентилятором, а час повного відкриття заслінки електроприводом доходить до двох хвилин.
- Рознесення моментів запуску електродвигунів. Асинхронні електродвигуни мають великі пускові струми. Так, компресори холодильних машин мають пускові струми, в 5-7 разів перевищують робочі (до 100А і більше). Якщо одночасно запустити вентилятори, холодильні машини і інші приводи, то через велике навантаження на електричну мережу будівлі сильно впаде напруга, і електродвигуни можуть не запуститися. Тому запуск електродвигунів, особливо великої потужності, необхідно розносити по часу.
- Попередній прогрів калорифера. Якщо включити кондиціонер, що не прогрів водяний калорифер, то при низькій температурі зовнішнього повітря може спрацювати захист від заморожування. Тому при включенні кондиціонера необхідно відкрити заслінки припливного повітря, відкрити триходовий клапан водяного калорифера і прогріти калорифер. Як правило, ця функція включається при температурі зовнішнього повітря нижче 12 ° С.

При відключенні системи слід враховувати:

- Затримку зупинки вентилятора припливного повітря в установках з електрокалорифером. Після зняття напруги з електрокалорифера слід охолоджувати його деякий час, не вимикаючи вентилятор припливного повітря. В іншому випадку нагрівальний елемент калорифера (трубчастий електричний нагрівач - ТЕН) може вийти з ладу.
- Затримку вимикання холодильної машини. При виключенні холодильної машини холодоагент зосередиться в найхолоднішому місці холодильного контуру, т. Е. В випарнику. При подальшому пуску можливий гідроудар. Тому перед вимиканням компресора, спочатку закривається клапан, який встановлюється перед випарником, а потім при досягненні тиску всмоктування 2,0-2,5бар, компресор вимикається. Разом з затримкою вимикання компресора виробляється затримка вимкнення припливного вентилятора.

Резервовані і доповнювані функції закладаються при роботі в схемі декількох однакових функціональних модулів (електрокалориферів, випарників, холодильних машин), коли в залежності від затребуваної продуктивності включаються один або кілька елементів. Для підвищення надійності встановлюються резервні вентилятори, електронагрівачі, холодильні машини. При цьому періодично (наприклад, через 100год) основний і резервний будуть різними функціями, вирівнюючи, таким чином, їх час напрацювання. Важливе значення мають функції програмного управління, такі як зміна режимів «зима-літо» та «день-ніч». Особливо актуальна реалізація цих функцій в сучасних умовах дефіциту енергетичних ресурсів. У найпростішому випадку ці функції передбачають або взагалі відключення ВКВ в певний момент часу, або зниження (підвищення) заданого значення регульованого параметра (наприклад, температури) в

залежності від періоду доби («день-ніч») або зміни теплових навантажень в приміщенні, що обслуговується. Більш ефективним, але і більш складним в реалізації, є програмне управління, що передбачає автоматичну зміну структури ВКВ і алгоритму її функціонування не тільки в традиційному режимі «зима-літо», а й в перехідних режимах.

Для вентиляції житлових приміщень досить звичайних вікон і витяжок. Управління системою побутової природної вентиляції може здійснюватися господарями приміщень без критичних ризиків. Однак сучасні вентиляційні системи промислових або громадських об'єктів, вироби яких розташовані по всій будівлі, вимагають іншого підходу - автоматизованої системи вентиляції приміщень, здатної підтримувати роботу в заданому режимі або змінювати кліматичні умови в приміщеннях без участі людини (на основі показників датчиків системи вентиляції).

Структура системи вентиляції в інтелектуальному будинку повинна бути сконструйована так, щоб відповідати нормам та при цьому бути енергоефективною. Енергоефективність вентиляції можна підвищити таким способом, як керування по споживанню. Для побудови такої системи треба вдало підібрані компоненти системи. Взагалі вона може складатися з різних, але є деякі такі, які буду встановлені майже у всіх системах вентиляції в інтелектуальних будівлях:

- Датчики (вуглекислого газу, температури тощо)
- Вентилятор з можливістю регулювання швидкості
- Заслінки або інші способи регулювання подачі повітря в приміщення
- Контролери
- Канали передачі даних
- Сервер для зберігання та обробки даних
- Програмне забезпечення

Наприклад рішення Swegon

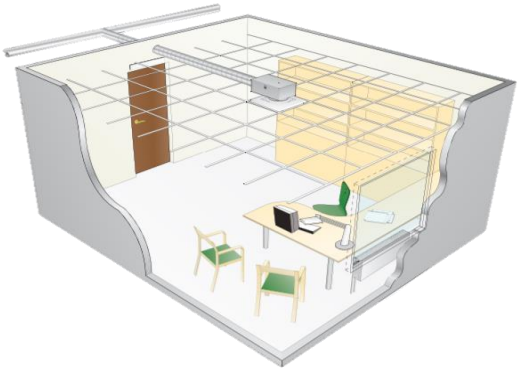

Рисунок 1.4 – Схематичне зображення рішення від Swegon

Це рішення надає вентиляція за потребою. Охолодження активними дифузорами припливного повітря. Управління радіаторами опалення.
Має такі компоненти:

- Активний дифузор припливного повітря ADAPT Colibri з вбудованими датчиками температури і присутності; с функцією вимірювання / управління витратою повітря з допомогою вбудованої заслінки з приводом
- Клапан управління радіатором опалення
- Протічний дифузор, наприклад, CGV / RGV, через який відпрацьоване повітря видаляється в коридор

Вбудований в дифузор датчик присутності подає сигнал на відкриття заслінки до її максимальної витрати при присутності людей в приміщенні і на її закриття до мінімальної витрати при відсутності, відповідно. Вбудований в дифузор датчик температури управляє охолодженням, плавно підвищуючи витрату повітря від мінімального до максимального, а також обігрівом, керуючи приводом клапана радіатора опалення. З метою енергозбереження в режимі відсутності допускаються відхилення температури від комфортної [10].

1.6 Постановка завдань магістерської дисертації

Проведення дослідження існуючих вентиляційних систем та способів їх керування. Проаналізувати існуючі датчики забруднення повітря. На основі цих даних зробити вибір необхідного датчика для системи. Розробити структуру систему контролю якості повітря. Провести аналіз пристроїв керування складовими частинами вентиляційної системи таких як: контролер заслінок, контролер приводу. Розробка структури для прийому і передачі даних. Розглянути та проаналізувати можливі системи збору даних. Також розглянути систему з використанням локальної мережі та глобальної мережі. Провести аналіз енергоефективності. Для визначення вимог щодо обміну даних в загальній системі якості повітря. Розробити програмне забезпечення для системи контролю якості повітря енергоефективної вентиляції.

1.7 Висновки

В даному розділі було розглянуте поняття інтелектуальних будівель. Також розглянуті які бувають забруднення повітря в будівлях та можливості енергозбереження за допомогою вентиляції. Проведено аналіз існуючих систем вентиляції та розглянуто структури систем вентиляції в інтелектуальних будівлях. Сформована постановка задач магістерської дисертації.

2 СТРУКТУРА СИСТЕМИ КОНТРОЛЮ ЯКОСТІ ПОВІТРЯ ДЛЯ ВЕНТИЛЯЦІЙНИХ СИСТЕМ

2.1 Датчики забруднення повітря

Перевірка якості атмосфери необхідна в приміщеннях і на вулиці - вона показує, наскільки характеристики відповідають безпечним умовам для здоров'я живих організмів, так як вони залежать від навколишнього середовища. Забруднення повітря навколишнього середовища можна вимірювати датчиками, що реагують на різні типи патогенних речовин. Більшість приладів для вимірювання забруднення повітря визначають концентрації поширених шкідливих домішок:

- Свинець - вражає нервову і серцево-судинну системи. Може привести до проблем з навчанням і порушень функцій нирок
- Діоксид сірки - сприяє погіршенню ґрунту і поверхневих вод
- Чадний газ - блокує надходження кисню до органів і тканин, що може привести до смерті
- Озон - вдихання O_3 провокує біль у грудях, кашель і подразнення горла. Високі концентрації речовини в приміщенні небезпечні, так як перехід в двохатомний стан супроводжується виділенням тепла, здатним викликати вибух або пожежу
- Дрібні тверді частинки - пил і сажа, розміром 10-25 мкм; при вдиханні вражають легені та інші внутрішні органи
- Ртуть - Випари елемента небезпечні для здоров'я
- Вуглекислий газ - підвищення концентрації в приміщенні веде до погіршення стану людини

Для здешевлення сучасні датчики часто фокусують на певному забруднювачі. Але діапазон розпізнавання загроз залежить від лежачих в основі пристрою механізмів.

Сучасні датчики мають декілька підходів до вимірювання. Вони діляться на:

- Випромінюючі. Зчитують зміни інфрачервоних або ультрафіолетових хвиль. Так, на початку 2019 го вчені створили масив золотих нанодисків, в 100 разів активніше сприймають атмосферні зміни, в порівнянні з існуючими датчиками. Вони реагували на ІЧ-лазер УФ-випромінюванням, інтенсивність якого залежала від концентрації цільових домішок.
- Електрохімічні. В основі лежать тверда речовина провідник. При нагріванні вона реагує з поступаючим повітрям в залежності від його складу.
- Біологічні. Дослідження, проведені восени 2018 го, показали, що мох чутливий до діоксиду сірки. Під дією речовини його листя згортаються. Інші рослини також допомагають стежити за станом повітря в реальному часі.
- Електроакустичні. Оцінюють зміни в частоті ультразвукових коливань. В основному застосовується для визначення рівня вуглекислого газу.

Вище приведена інформація, яка стосується більшості датчиків [6]. Для системи вентиляції можуть використовуватися декілька датчиків, які встановлюються залежно від потреб та конкретної ситуації. Система вентиляції, яка представлена в цій роботі, використовує датчик вуглекислого газу.

Датчики CO2 призначені для вимірювання концентрації і контролю граничних значень вуглекислого газу в діапазоні від 0 до 100%. Дані пристрої застосовуються в системах охоронних сигналізаціях, автоматичних системах вентиляції, контролюючих якість повітря, для визначення вмісту вуглекислоти в повітряному потоці, а також рівня газу в приміщенні і на свіжому повітрі. У нормі концентрація CO2 повинна знаходитися в межах 0,04%. Зрозуміло, що таку його кількість постійно підтримувати в

приміщенні практично неможливо. При цьому навіть незначне збільшення концентрації вуглекислого газу може згубно позначитися не тільки на працездатності, а й на загальний стан людини, приводячи до негативних змін в складі крові, зниження pH, ацидозу, збільшення концентрації бікарбонату і кисневого голодування. Датчик, який визначає процентний вміст вуглекислоти в навколишньому повітрі, може допомогти своєчасно запобігти розвитку цих негативних наслідків і швидко нормалізувати порушений газообмін.

Але існує декілька типів датчиків вуглекислого газу і для кожної ситуації підходить той чи інший. Основні типи датчиків CO_2 на ринку поділяються на три категорії:

- Електрохімічні датчики
- Напівпровідникові датчики оксиду металу
- Недисперсні інфрачервоні датчики

Кожен тип датчика має свої сильні та слабкі сторони, про які слід пам’ятати перед тим, як приймати рішення щодо того, який саме використовувати.

2.1.1 Електрохімічні датчики

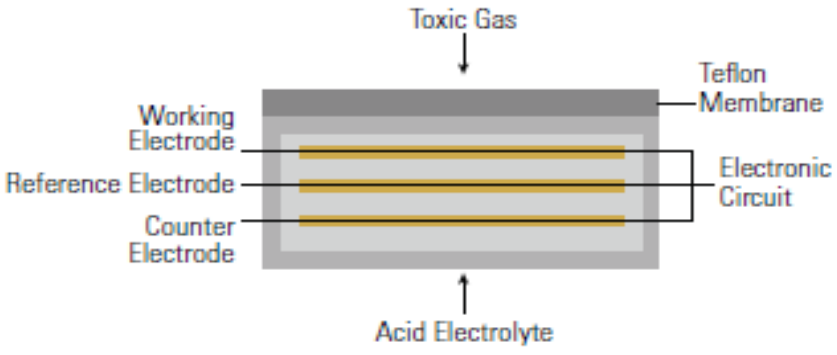

Рисунок 2.1 – Структура електрохімічного датчику

Електрохімічні датчики вуглекислого газу вимірюють електричний струм або провідність, щоб визначити, скільки CO_2 знаходиться у повітрі. Коли частинки вуглекислого газу потрапляють в датчик, то проходить хімічна реакці. Коли ця реакція відбувається, датчик фіксує зміни у електричному струмі. Потім датчик буде використовувати тип і величину електричних змін, щоб визначити, яка концентрація вуглекислого знаходиться у повітрі.

Переваги:

- Менш сприйнятливий до вологості та перепадів температури ніж інші види датчиків.

Недоліки:

- Інші речовини можуть змінити показання
- Мають не такий великий термін експлуатації, як недисперсні інфрачервоні датчики
- Датчик може «дрейфувати» або втрачати точність

2.1.2 Напівпровідникові датчики

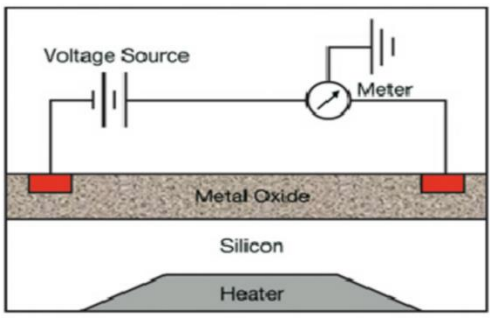

Рисунок 2.2 – Структура напівпровідникового датчику оксиду металу

Напівпровідникові датчики оксиду металу вуглекислого газу використовують питомий опір для перевірки кількості газу в повітрі. Такий датчик має металеву смужку або плівку, які піддаються впливу повітря, яке ми хочемо перевірити. Через цю смугу проходить електричний ток. У міру контакту цільового газу із плівкою він буде взаємодіяти з металом і змінювати хімічний склад або за допомогою реакції відновлення або окислення. Коли це станеться, опір, або провідність металу будуть змінені. Вид зміни опору, незалежно від того, збільшується чи зменшується, і величина цієї зміни визначає концентрацію цільового газу. Залежно від того, що це за метал, різні гази будуть реагувати на смугу.

Переваги:

- Дуже простий дизайн робить їх простими у використанні

Недоліки:

- Вологість та температура може впливати на точність вимірювання
- Зазвичай використовується при більш високих, менш поширених концентраціях CO_2 (> 2000 ppm)
- Інші речовини в повітрі можуть змінювати показання

2.1.3 Недисперсні інфрачервоні датчики

Кожен елемент на Землі поглинає світло у певних діапазонах. Наприклад, морква відображатиме помаранчеве світло, але поглинатиме всі інші кольори. Оскільки світло поглинається, воно не має можливості потрапити до наших очей, або інших приборів. Тому недисперсні інфрачервоні датчик працює за підходом, який описаний нижче.

Принцип роботи даного сенсорного пристрою заснований на зміні інтенсивності інфрачервоного випромінювання до і після поглинання його

вуглекислим газом в діапазоні 1-15 мкм. Далі вимірюється кількість світла, що пройшло через світлофільтр і поглиненого вуглекислотою. Після порівняння з показниками потоку світлового випромінювання, що пройшов через оптичний пристрій, прилад визначає різницю і видає показник концентрації вуглекислого газу. Недисперсний інфрачервоний метод детектування відрізняється високою стабільністю, хорошою вибірковістю і не залежить від змісту кисню, який знаходиться у повітрі [7].

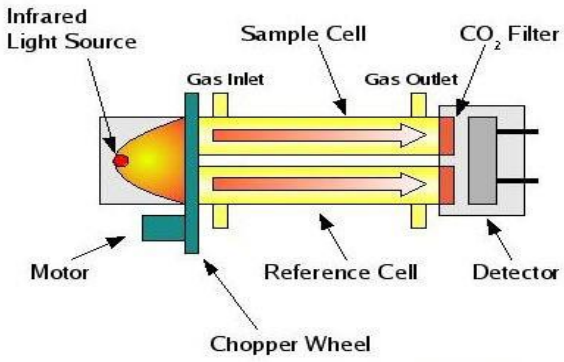

Рисунок 2.3 – Структура недисперсного інфрачервоного датчику

Переваги:

- Довготривалий термін експлуатації (деякі можуть працювати більше десяти років)
- Інші речовини не заважають показанням
- Добре працює в діапазонах CO_2 близько 1000 ppm

Недоліки:

- Вологість та температура може впливати на точність вимірювання

Для використання в системі вентиляції було обрано недисперсний інфрачервоний датчик. Насьогодні більшість існуючих датчиків такого типу йдуть ще з датчиком температури та вологості. Ми можемо отримати і ці дані, але першочергово вони відіграють компенсаційну роль. За допомогою

даних с цих датчиків відбувається корекція даних концентрації вуглекислого газу у повітрі.

Недисперсні газоаналізатори, що застосовуються для визначення концентрації вуглекислого газу в повітрі, комплектуються світлофільтрами або спеціальними приймачами. Крім світлового фільтру, пристрої оснащуються робочою камерою. Подається туди повітря, яке далі потрапляє в спеціальний приймач. Як джерело випромінювання може виступати світлодіод, нагріта спіраль або інфрачервоний лазер. Залежно від рівня концентрації вуглекислий газ поглинає частину світлових променів, пропорційно змінюючи реєстрований сигнал. При використанні джерела немонохроматичного випромінювання для отримання достовірного результату застосовується селективний приймач. Крім селективних приймачів в інфрачервоних датчиках CO2 можуть використовуватися термобатареи, болометри або напівпровідникові елементи. У даній ситуації прилади оснащуються газовими або інтерференційними фільтрами. Також деякі виробники аналізаторів газу при виготовленні приладів застосовують збірний фільтр. Перший пропускає встановлену інфрачервону смугу, а другий - ні. Позаду фільтрувального елемента розташовується збірний детектор і пов'язаний з ним обчислювальний пристрій, яке формує неузгодженість сигналів. Як матеріал, що використовується для виготовлення датчика, застосовують стандартний полікарбонат або акрилнітрил-бутадієн-стирол. Ці легкі, і в той же час неймовірно міцні пластичні сплави легко піддаються обробці, мають відмінні ізоляційні властивості і високу стійкість до хімічного впливу. Газоаналізатори, призначені для застосування під відкритим небом, виготовляються з полікарбонату. У той же час прилади з більш низькою стійкістю до ударів і діапазоном робочих температур застосовуються тільки всередині приміщень. Дисперсійні аналізатори передбачають використання однохвильового випромінювання, отриманого за допомогою монохроматографа. Що ж

стосується недисперсних пристроїв, то в їх роботі застосовується немонохроматичним випромінювання, послідовно проходить через світлофільтр і робочу камеру, яка містить аналізовану суміш (в даному випадку повітря). Присутній в ньому вуглекислий газ поглинає частину випромінювання (яку - залежить від концентрації). Як наслідок, відбувається пропорційна зміна реєстрованого сигналу.

Кожен CO2 датчик виконує свою функцію, а саме вимірювання вуглекислого газу. Але кожен датчик може відрізнятися характеристиками. Кожен датчик має наступні характеристики:

- Напруга електроживлення / Потужність
- Діапазон вимірювання
- Точність вимірювання
- Час відгуку
- Час входження в режим роботи щоразу під час увімкнення
- Вихід (дискретний або аналоговий та який протокол)
- Експлуатаційні умови
- Розмір

Розглянемо найбільш поширені датчики і проаналізуємо їх характеристики:

SCD30

Рисунок 2.4 – датчик SCD30

Таблиця 2.1 – Характеристики датчика SCD30

Характеристика	Значення
Діапазон вимірювання	0 - 40000 ppm

Точність	± (30 ppm + 3%)
Роздільна здатність	10 ppm
Час відгуку	20 с
Робоча напруга	3.3 - 5.5 В

MG-811

Рисунок 2.5 – датчик MG-811

Таблиця 2.2 – Характеристики датчика MG-811

Характеристика	Значення
Діапазон вимірювання	350 - 10000 ppm
Точність	± (30 ppm + 5%)
Роздільна здатність	1 ppm
Час відгуку	30 с
Робоча напруга	5В

MH-Z19B

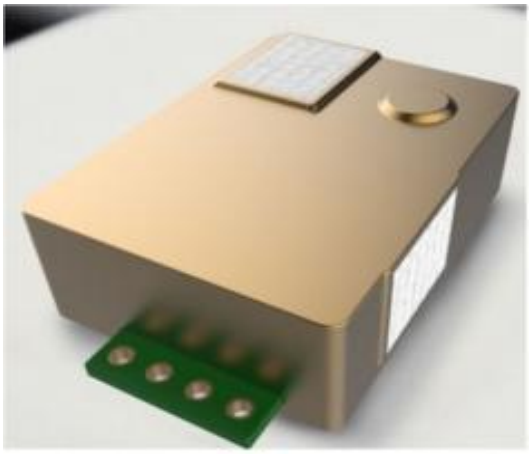

Рисунок 2.6 – датчик MH-Z19B

Таблиця 2.3 – Характеристики датчика MH-Z19B

Характеристика	Значення
Діапазон вимірювання	0 - 5000 ppm
Точність	± (50 ppm + 5%)
Роздільна здатність	1 ppm
Час відгуку	30 с
Робоча напруга	3.6 - 5.5 В

Для даної системи контролю якості повітря і керування вентиляцією у інтелектуальних будівлях було обрано саме MH-Z19B. Він найкраще підходить для даної цілі тому, що його діапазон відповідає всім вимогам системи, а також не завеликий, що робить його більш доступним з фінансової точки зору. Також він має гарну роздільну здатність та задовільний час відгуку. Плюсом цього датчика ще являється те, що він має два виходу для отримання даних, як аналоговий, так і цифровий за допомогою універсального асинхронного передавача, більш відомого як UART.

2.2 Контролери для управління заслінками і електроприводами вентиляторів

Для створення регульованого електроприводу доступна широка номенклатура мікроконтролерів від різних виробників: Texas Instruments, Freescale, Microchip, Atmel, Intel, Fuji, Infineon і ін. Контролери різняться

розрядністю (8 ... 32-бітові), архітектурою (RISK, DSP, CISC), продуктивністю , наявністю спеціалізованого блоку управління ШІМ деяких інших характеристик. Практика показала успішність застосування як спеціалізованих мікроконтролерів електроприводу, так і мікроконтролерів загального призначення.

Однак каменем спотикання на шляху розробки регульованого електроприводу є розробка спеціалізованого програмного забезпечення, що вимагає великих знань теорії електроприводу. Застосування готових бібліотек, набору програм і рішень від виробників мікроконтролерів в якійсь мірі спростило завдання розробника і програміста. Але використання в серійному виробництві показало обмеженість готових рішень від виробника, складність внесення змін і адаптації електроприводу до вимог замовника. Внесення змін в існуючу програму перетворюється для розробника в «підстановку милиць» і незмінно закінчується глибокою переробкою вже існуючих програм. Невисока якість реалізації прикладів векторного управління також обмежує застосування електроприводу.

Поява в останнє десятиліття економічно і енергетично привабливих синхронних електродвигунів з постійними магнітами призвело до загострення перерахованих вище проблем, оскільки такі двигуни можуть працювати тільки з зовнішньою системою управління. З успішних апаратних реалізацій управління приводом можна виділити мікроконтролер MC3PHAC компанії Freescale Semiconductor зі скалярним керуванням асинхронним електроприводом. Незважаючи на те, що MC3PHAC випущений досить давно, даний контролер успішно застосовується в рішеннях типових задач з невисокими вимогами до якості управління - насоси, вентилятори і т.д.

Аналіз ринку електроприводу і напрямків його розвитку привів International Rectifier до розробки спеціалізованих конфігуруються контролерів для синхронних машин з постійними магнітами і трифазним синусоїдальним струмом з бездатчикового управлінням. У 2004 році компанія представила

спеціалізовану платформу для електроприводу - iMotion. Ядром платформи стали конфігуруються контролери електроприводу. До складу платформи також увійшли окремі компоненти силової електроніки - інтелектуальні силові модулі, датчики струму, драйвери. Важливою особливістю конфігуруються контролерів International Rectifier є те, що вперше векторне керування електроприводом стало складовою частиною контролера.

Перше покоління конфігуруються контролерів представлено контролерами IRMCK201 і IRMCK203, друге покоління - суттєво більш різноманітною за функціоналом серією IRMCK / IRMCF3xx. Контролери IRMCKxxx орієнтовані на застосування в складі електроприводу трифазних синхронних машин з синусоїдальним струмом. Для застосування з асинхронним приводом призначені рішення на IRMCK201 і IRMCK341. Апаратна реалізація векторного управління дозволила отримати поєднання високої швидкодії, точності, стійкості алгоритмів управління та низької вартості контролера. Продуктивність IRMCK201 і IRMCK203 в чотири рази перевищує продуктивність спеціалізованих DSP для електроприводу.

Таблиця 2.4 – Основні параметри IRMCK201 і IRMCK203

Тактова частота генератора, МГц	33,3
Внутрішня тактова частота контролера, МГц	133,3
Час розрахунку замкнутого контуру управління, мкс	не більше 6
Смуга пропускання замкнутого контуру по току (-3 dB), кГц	5,5
Вихідна частота ШІМ, кГц	до 83,3
Тактова частота SPI-інтерфейсу, МГц	до 8

Обидва мікроконтролера підтримують даний тип інтерфейсів: RS232C / RS422, SPI (до 8 МГц), 8-бітний паралельний інтерфейс. Завантаження налаштувань конфігурації контролерів здійснюється з зовнішньої EEPROM. В обох контролерах реалізований апаратний модуль просторової векторної модуляції трифазного ШІМ. Вимірювання струму проходить по двох фаз, з відновленням струму в третій фазі. Модуль вимірювання струму в першу

чергу орієнтований на пряме підключення мікросхем вимірювання струму IR2175 і подібних до них.

У IRMCK201 і IRMCK203 є і відмінності.

IRMCK201 є найбільш функціональним контролером: він може працювати як в замкнутій по швидкості системі управління, так і в розімкнутої. Даний контролер може застосовуватися як в синхронному електроприводі з постійними магнітами, так і в асинхронному приводі. Замкнута система управління дозволяє отримати високоточний привод з діапазоном регулювання до 1: 5000. IRMCK201 знаходить застосування в різних системах, сервоприводах роботів-маніпуляторів, високоточному електроприводі.

У IRMCK203 реалізована розімкнута за швидкістю система управління, орієнтована на застосування тільки з синхронними трифазними електродвигунами з постійними магнітами. Проте, векторне управління дозволило реалізувати стійке регулювання швидкості в діапазоні 1:50.

Висока продуктивність контролерів дозволяє реалізувати як привід для типових застосувань (промисловий привід, насосне обладнання, вентиляція, привід засувок і т.п.), так і спеціалізований високооборотний привід (до 100000 об / хв). Приклади успішного застосування в високооборотному приводі: турбомолекулярні вакуумні насоси, високопродуктивний електроінструмент, бормашини, турбодетандери.

Конфігуруванні контролери серії IRMCK2xx і IRMCK / IRMCF3xx мають ряд важливих відмінностей. Головні з них - поява у другому поколінні вдосконаленого ядра мікроконтролера 8051, модуля MCETM (Motion Control Engine) і версії з вбудованою флеш-пам'яттю програм. Друге покоління конфігурування контролерів пішло по шляху реалізації бездатчикового векторного управління. Тут компанія International Rectifier відмовилася від ідеї одного універсального контролера, випустивши кілька типів контролерів з конфігурацією, орієнтованої на додатки [8].

Таблиця 2.5 – Типи контролерів з конфігурацією, орієнтованої на додатки

Назва (Корпус)	Функції	Аналоговий ввід/вивід	Цифрові порти вводу/виводу	Інтерфейси
IRMCF312 IRMCK312 (QFP100)	2 двигуна 1 ККП	12-бітний АЦП 11 каналів АЦП POR UVLO Аналоговий таймер	36 портів в/в 1 порт захвату 4 таймера	RS232 x 2 I^2C/SPI
IRMCF311 IRMCK311 (QFP64)	2 двигуна 1 ККП	12-бітний АЦП 6 каналів АЦП POR UVLO Аналоговий таймер	14 портів в/в 1 порт захвату 4 таймера	RS232 x 2 I^2C/SPI
IRMCF343 IRMCK343 (QFP64)	2 двигуна 1 ККП	12-бітний АЦП 5 каналів АЦП POR UVLO Аналоговий таймер	23 порту в/в 1 порт захвату 4 таймера	RS232 I^2C/SPI
IRMCF341 IRMCK341 (QFP64)	1 двигун	12-бітний АЦП 8 каналів АЦП POR UVLO Аналоговий таймер	24 порту в/в 1 порт захвату 4 таймера	RS232 I^2C/SPI
IRMCF371 IRMCK371 (QFP48)	1 двигун	12-бітний АЦП 4 каналу АЦП POR UVLO Аналоговий таймер	13 портів в/в 1 порт захвату 4 таймера	RS232 I^2C/SPI
ККП — коректор коефіцієнта потужності IRMCF — програмна пам'ять 48 кбайт, пам'ять даних 8 кбайт IRMCK — програмна пам'ять 56 кбайт, пам'ять даних 8 кбайт POR — скидання з живлення; UVLO — захист від низької напруги;				

Siemens S7-200 Програмований контролер

Рисунок 2.7 – програмований контролер Siemens S7-200

Мікроконтролери SIMATIC S7-200 призначені для вирішення завдань управління і регулювання в невеликих системах автоматизації. При цьому, SIMATIC S7-200 дозволяють створювати як автономні системи управління, так і системи управління, що працюють в загальній інформаційній мережі. Область застосування контролерів SIMATIC S7-200 виключно широка і

простягається від найпростіших завдань автоматизації, для вирішення яких в минулому використовувалися прості реле і контактори, до завдань комплексної автоматизації. SIMATIC S7-200 все більш інтенсивно використовується при створенні таких систем управління, для яких в минулому з міркувань економії необхідно було розробляти спеціальні електронні модулі.

Мікропроцесори Siemens S7-200 мають велику кількість:

- Базових операцій: логічні інструкції, інструкції адресації результату, збереження даних, управління таймерами і лічильниками, завантаження, передачі, порівняння, зсувних операцій, формування доповнень, виклику підпрограм (з передачею локальних змінних).
- Інтегрованих комунікаційних функцій: читання (NETR) і записи (NETW) інформації в мережу, підтримки вільно програмованого порту (Transmit XMT, Receive RCV).
- Функцій розширеного набору команд: інструкції управління широтно-імпульсною модуляцією, генераторами імпульсів, виконанням арифметичних функцій і операцій з плаваючою комою, роботою ПІД регуляторів, функціями переходів і циклів, перетворення кодів та інші.
- Лічильники: зручний набір функцій в поєднанні з вбудованими швидкісними лічильниками істотно розширюють можливий спектр областей застосування контролера.

Обробка переривань:

- Використання входів апаратних переривань, фіксують появу імпульсних сигналів (по наростаючому чи спадающему фронту) і дозволяють істотно знизити час реакції контролера на запити, що надходять.

- Тимчасові переривання, періодичність повторення яких може здаватися з кроком в 1мс в діапазоні від 1 до 255 мс.
- Переривання від лічильників: можуть формуватися в моменти досягнення заданого значення або зміни напрямку рахунку.
- Комунікаційні переривання: забезпечують підвищення ефективності зв'язку з периферійним обладнанням, наприклад, з принтером або сканером штрих-кодів.
- Пряме сканування входів і виходів, вироблене незалежно від циклу виконання програми.
- Безпосереднє опитування входів і управління виходами: опитування входів і управління станом виходів може виконуватися незалежно від циклу виконання програми. Це дозволяє знизити час реакції на переривання і час формування відповідних вихідних сигналів.

Реєстрація даних: Керована або періодична реєстрація даних в модулі EEPROM пам'яті, наприклад, статистичних даних, повідомлень про помилки і т.д. Опціонально зареєстровані дані можуть доповнюватися відмітками дати і часу. Реєстраційний файл в будь-який момент може бути переданий в STEP 7-Micro / WIN з використанням S7 Explorer.

Управління рецептами: Рецептури завантажуються разом з проектом STEP 7-Micro / WIN. Для оптимізації використання пам'яті рецепти зберігаються в модулі EEPROM пам'яті [9]. Рецепти можуть модифікуватися і доповнюватися в інтерактивному режимі.

Функціональні особливості:

- Програмовані контролери, що відрізняються максимумом ефективності при мінімумі витрат.
- Простота монтажу, програмування і обслуговування.
- Рішення як простих, так і комплексних завдань автоматизації.

- Можливість застосування в вигляді автономних систем або в якості інтелектуальних ведених пристроїв систем розподіленого вводу-виводу.
- Можливість використання в сферах, де застосування контролерів раніше вважалося економічно недоцільним.
- Робота в реальному масштабі часу і потужні комунікаційні можливості (PPI, MPI, Industrial Ethernet, PROFIBUS-DP, AS інтерфейс, модемний зв'язок).
- Компактні розміри, можливість установки в обмежених обсягах.

ADAPT Damper - це пристрій керування змінною витратою повітря системи Swegon WISE. Положення заслінки визначає коректний витрата повітря, виходячи із заданих значень: витрата повітря для режиму відсутності, хв. і макс. витрати. Пристрій ADAPT Damper оснащено контролером для актуальних заданих значень і функцій управління, а також вбудованим датчиком температури ПВ або ОВ. Сигнал системи диспетчеризації (BMS) може повністю відкрити або закрити заслінку (аварійне становище). Закриває заслінка пристроїв круглого перетину відповідає класу щільності 3. Матеріали і покриття Пристрій ADAPT Damper виконано з оцинкованої листової сталі з компонентами електроніки, а також з пластика і гуми. Коробка підключень - з ABS-пластика

Приклад використання:

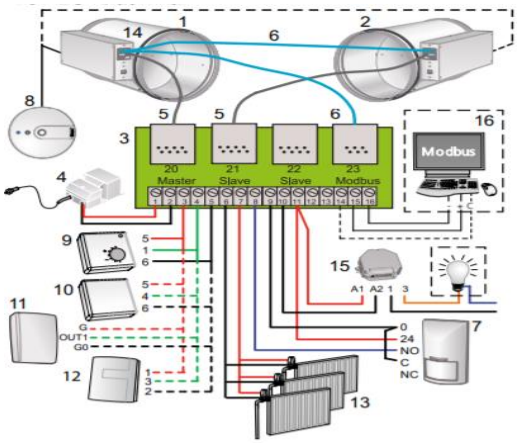

Рисунок 2.8 – Приклад використання системи Swegon

Заслінка монтується в повітроводи ПВ і ВВ класної кімнати або коридору. Заслінка master монтується в повітропровід ВВ, і витрата повітря класної кімнати забезпечує задану температуру і якість повітря в ній. Використання датчика присутності дозволяє додатково працювати з іншими показниками витрати і температури повітря під час відсутності людей. Крім того, освітлення може також управлятися від датчика присутності, що додатково заощаджує енергію та гроші. Пристрій ADAPT Damper змінює своє положення між хв. і макс. витратами повітря в залежності від заданих значень і інформації від датчиків (по перепаду тиску системи). Макс. витрата повітря вибирається при заслінки, відкритої не більше ніж на 80%. Тиск перед заслінкою потрібно враховувати з точки зору рівня шуму (див. Діаграми вибору), воно не повинно перевищувати 250 Па. Кожна зона повинна управлятися по тиску, що виконується в малих системах за допомогою вентагрегатами, а в великих - за допомогою зональної заслінки CONTROL Zone (див. Також теоретичну частину опису WISE). Системи тільки з заслінкою ПВ повинні комплектуватися датчиком DETECT Та або DETECT SME.

Конференц-зал зі змінною подачею повітря:

Рисунок 2.9 – Схематичний рисунок конференц-залу з вентиляцією за споживанням

Вар. 1 Управління тільки по температурі по вбудованому в заслінку ВВ датчику температури

Вар. 2 В комплекті з датчиком присутності можливість задати значення витрати повітря і температури в умовах відсутності людей

Вар. 3 В комплекті з датчиком управління якістю повітря.

Налагодження ADAPT Damper стандартно проходить наладку на заводі. Контроль актуальних значень витрати повітря вручну, а також можливу зміну заданих значень, проводиться за допомогою TUNE Adapt, що підключається в наявний контакт пристрою ADAPT Damper. Обслуговування ADAPT Damper не вимагає обслуговування. Чистка за допомогою пилососа. При необхідності чищення повітропроводів і відсутності санітарних кришок в них, ADAPT Damper демонтується. Заміна важливих компонентів може здійснюватися без необхідності демонтажу всієї заслінки [10].

2.3 Організація системи збору даних (з використанням локальної або глобальної мережі)

Система збору даних може бути організована з використанням, як локальної мережа, так і глобальної. Передача даних від датчиків може відбуватися за допомогою кабелю або Wi-Fi.

Плюси використання дротового з’єднання полягає у тому, що підключення є більш стабільним та швидкість передачі даних може досягати великих значень. Але об’єм даних, які передаються від датчиків не перевищують кілобайти. Тому було обрано бездротовий спосіб передачі даних. Окрім переваги у відсутності проводів, що надає змогу проводити менше робіт пов’язаних на побудові інфраструктури, з’являється можливість з легкістю змінювати положення датчика або якось його модифікувати. Також роз’ємів у маршрутизатора не так багато, як може бути датчиків, що призвело би до збільшення витрат на підтримку такого підходу. А до точки бездротового з’єднання можна підключати сотні пристроїв.

Локальна обчислювальна мережа (ЛОМ, локальна мережа, англ. Local Area Network, LAN) - комп'ютерна мережа, що покриває зазвичай відносно невелику територію або невелику групу будівель (будинок, офіс, фірму, інститут).

Комп'ютери можуть з'єднуватися між собою, використовуючи різні середовища доступу: мідні провідники (кручена пари), оптичні провідники (оптичні кабелі) і через радіоканал (бездротові технології). Провідні, оптичні зв'язки встановлюються через Ethernet і інші засоби. Окрема локальна обчислювальна мережа може мати зв'язок з іншими локальними мережами через шлюзи, а також бути частиною глобальної обчислювальної мережі (наприклад, Інтернет) або мати підключення до неї.

Використання локальної мережі доцільно тоді, коли наш серверна частина програмного забезпечення для обробки даних знаходиться у самій будівлі. Також це набагато збільшує безпеку системи тим, що неможливо буде нашкодити процесу її роботи за допомогою віддаленого підключення.

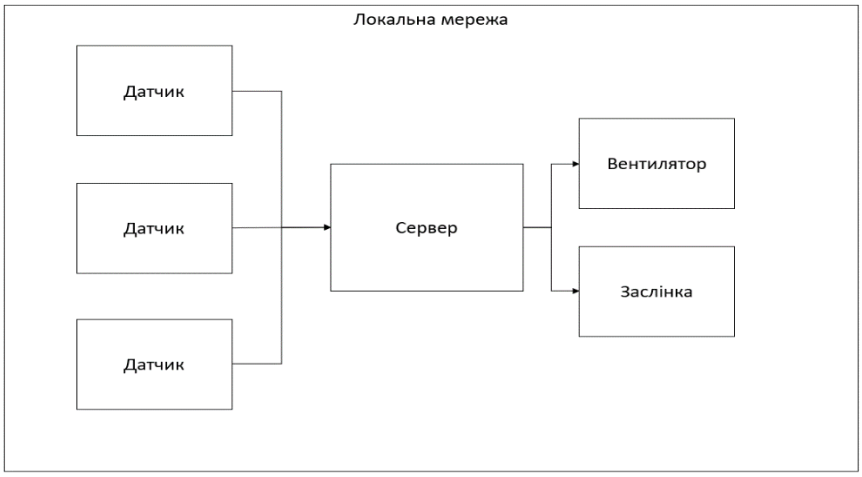

Рисунок 2.10 – Структура системи з використання локальної мережі

Також є можливість використання глобальної мережі. У такому випадку сервер може бути один або це навіть може бути централізований сервіс, який би керував вентиляційними системами багатьох будівель. У такому випадку не потрібно встановлювати таку систему у кожну будівлю, а треба лише підключити датчики, заслінки та вентилятори і налаштувати маршрути передачі даних та контроль доступу.

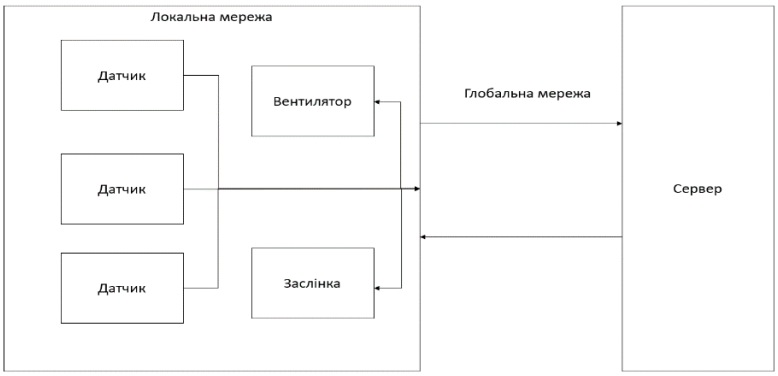

Рисунок 2.11 – Структура системи з використання глобальної мережі

2.4 Оцінка ефективності використання запропонованого рішення

Система вентиляції за потребою відповідає найжорсткішим вимогам енергоефективності та в порівнянні з системою постійної витрати повітря

дозволяє заощадити до 80% енергоспоживання вентиляторів. В роботі запропонованої структури системи для забезпечення енергоефективного керування вентиляцією з підтриманням концентрації вуглекислого газу в повітрі в заданих межах. Показано, що використання недисперсних інфрачервоних датчиків є найбільш доцільним в такій системі.

Для проведення оцінки ефективності запропонованої системи розглянемо енергетичні показники електроприводу з використанням різних підходів до керування та за умови вентиляторного характеру його навантаження.

Оцінку енергетичної ефективності будемо проводити розглянувши основні види втрати енергії, що присутні в усіх складових частинах нашої системи.

Вентилятор характеризується коефіцієнтом корисної дії який залежить від конструкції вентилятора та конструкції вентиляційної системи. Цей показник є постійним при зміні швидкості і тому в запропонованій системі його можна не приймати до уваги.

Втрати енергії в перетворювачі частоти визначається втратами на перемикаючих елементах. Вони складаються з комутаційний та активних втрат. Активні втрати залежать від квадрату величини струму що через них протікає, а комутаційні втрати - від величини струму, що протікає і частоти перемикань напівпровідникових транзисторів. Враховуючи те, що частота перемикань при роботі перетворювача частоти є постійною, то для мінімізації втрат перетворювача частоти доцільно мінімізувати струм.

Втрати в асинхронному електричному двигуні згідно з [11], визначаються наступним виразом

$$\sum P_n = P_{e1} + P_{e2} + P_{ст} + P_{мех} + P_{дод} + P_{гар}, \text{ де}$$

Pe1, Pe2 - електричні втрати в обмотках статора і ротора; Рст - втрати в сталі статора; Рмех - механічні втрати; Рдод - додаткові втрати; Ргар - гармонічні втрати.

Електричні втрати в обмотках обумовлені протіканням по ним електричного струму. Втрати в сталі статора складаються з втрат на гістерезис і втрат від вихрових струмів. Додаткові втрати включають в себе втрати від вихрових струмів в обмотках, пульсації втрати в зубцях, втрати в сталі ротора, втрати від потоків розсіювання, пульсацій індукції в повітряному зазорі і ін. Гармонічні втрати виникають від вищих гармонік при несинусоїдної форми напруги і струму статора. Механічні втрати в електричній машині і механічної передачі обумовлені взаємодією рухомих частин з нерухомими частинами, приєднаними деталями та навколишнім середовищем.

Необхідно враховувати, що механічні втрати визначаються конструкцією електричної машини, типом механічної передачі, якщо вона присутня і частотою обертання, і не залежать від використовуваного підходу до керування. Гармонічні втрати теж не залежать від підходу до керування, при цьому їх зменшення вимагає підвищення частоти перемикань силових приладів перетворювача частоти, що призводить до збільшення комутаційних втрат. Додаткові втрати залежать від квадрата струму статора і можуть досягати 1-2% загальної потужності. Тому можна зробити висновок, що для аналізу енергетичної ефективності використання запропонованої системи доцільно розглядати лише втрати енергії в обмотках двигуна та втрати сталі статора.

Проаналізуємо з використанням підходу запропонованого в [11] механічну енергію, що споживається електричним двигуном, а також рівень втрат в електричному двигуні при роботі вентилятора с постійними швидкостями.

Аналіз будемо проводити для двигуна 4А160М4У3 з параметрами: R_s = 0.264 Ом, R_r = 0.151 Ом, L_s = 1.7 мГн, L_r = 2.6 мГн, L_M = 88 мГн, p = 2, $\Delta s_{ном}$=1.1 Гц, n_c=1500 об/хв, $P_{ном}$ = 18.5 кВт. При цьому будемо вважати що номінальній швидкості обертання відповідає момент рівний 100 Н/м

вентиляторного навантаження. Враховуючи те, що момент пропорційний квадрату швидкості аналіз будемо проводити для швидкостей обертання від 1500 об/хв до 500 об/хв. При цьому розглянемо два підходи: в одному з яких забезпечується максимізація відношення момент струм, а в іншому забезпечується постійний рівень намагнічування в асинхронному двигуні. Результати розрахунку наведені в таблиці 2.3.

Таблиця 2.6 – Результати розрахунку

N, об/хв	M, Н/м	Рмех, Вт	При мінімальному струму статору			При постійному струму намагнічування		
			Рст, Вт	Рел, Вт	∑, Вт	Рст, Вт	Рел, Вт	∑, Вт
1500	100	15708	326	887	1213	249	1119	1368
1400	87	13195	320	500	820	224	830	1054
1300	75	10891	280	312	592	200	603	803
1200	64	8796,5	252	409	661	176	512	688
1100	54	6911,5	200	301	501	154	380	534
1000	44	5236	160	250	410	133	283	416
900	36	3769,9	115	181	296	113	199	312
800	28	2513,3	80	109	189	94	100	194
700	22	1466,1	64	82	146	76	83	159
600	16	1256,6	40	60	100	60	65	125
500	11	523,6	20	40	60	45	49	94

По результатам побудуємо графіки залежності механічної потужності (рисунок 2.12) від частоти обертання та рівня втрат енергії від частоти обертання (рисунок 2.13).

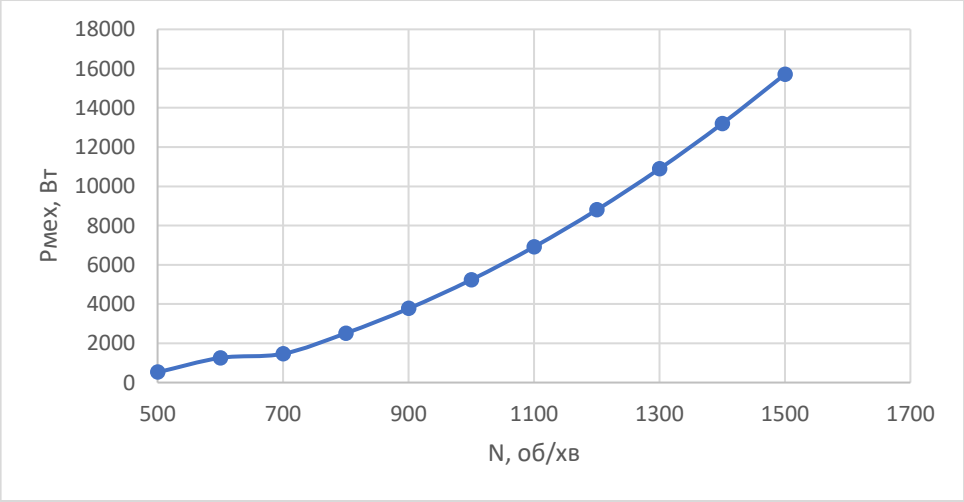

Рисунок 2.12 – Залежність механічної потужності від швидкості обертання

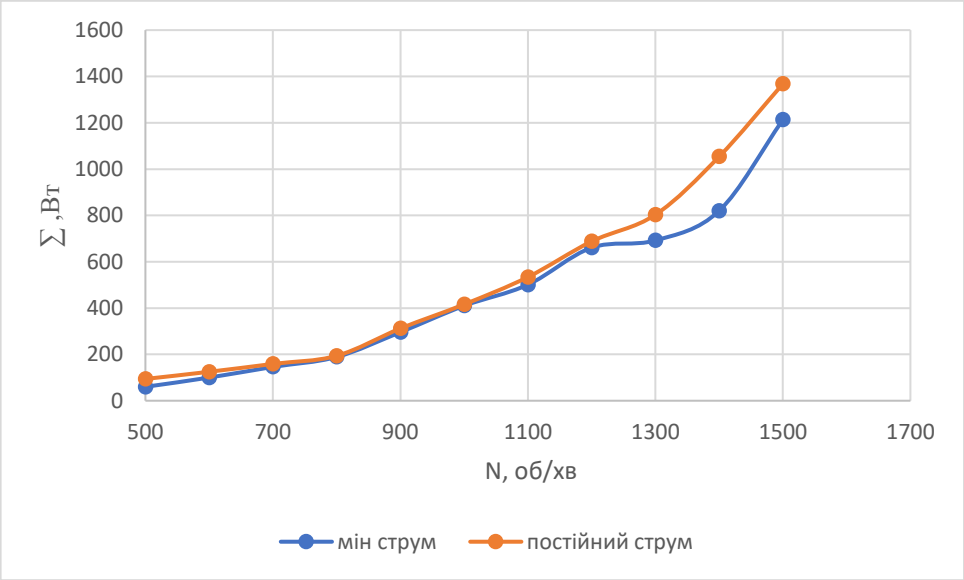

Рисунок 2.13 – Залежність суми втрат від швидкості обертання

Виходячи з отриманих даних можна зробити висновок, що при зменшенні обсягу повітря, що подається в вентиляцію вдається суттєво знизити споживану двигуном енергію. Так при зменшенні подачі вентиляцією повітря в 3 рази зменшення механічної потужності в 30 разів. Не зважаючи на те, що ККД двигуни при цьому зменшується, абсолютне значення сумарних втрат зменшується в 20 разів. Виходячи з наведених графіків можна зробити висновок, що використання підходу при якому забезпечується мінімізація струму статора є кращою аніж забезпечення постійного рівня намагнічування електричного двигуна.

2.5 Формування вимог до програмної частини

Розробка програмної частини невід'ємний процес створення вентиляції для інтелектуальної будівлі. Саме за рахунок сукупності цих компонентів і злагодженої роботи стає можливим досягнути підвищенню енергоефективності вентиляційної системи.

Перш за все програмна система буде поділятися на такі частини:

- Апаратно-програмна частина

- Серверна частина
- Клієнтська частина
- База даних

Апаратно-програмна частина повинна працювати на одноплатному комп’ютері Raspberry Pi, тому для цього програмна частина буде написана на мові програмування Python. Серверна частина повинна передавати дані на клієнтську у реальному часі, щоб дані були актуальні та отримувати з датчику. Для передачі даних у реальному часі до клієнтської частини програма повинна бути сумісною з протоколом WebSocket. Також для більш тісної інтеграції клієнтська частина повинна бути написана на тій же мові програмування, що і серверна частина, щоб можна було використовувати спільні компоненти про розробці. Також для зберігання даних вимірювання повинна використовуватися реляційна база даних.

2.6 Висновки

В даному розділі були розглянуті датчики забруднення повітря і більш детально проаналізовані датчики вуглекислого газу та їх різновиди, а саме електрохімічні, напівпровідникові та недисперсні інфрачервоні датчики. Розглянуті контролери для управління заслінками та електроприводами. Було проведено огляд різних підходів до організації систему збору даних. Сформовано вимоги до програмної частини.

3 РОЗРОБКА ПРОГРАМНИХ КОМПОНЕНТІВ

3.1 Розробка програмного забезпечення контролерів для контролю параметрів повітря

Є декілька платформ на яких доцільно було б створити програмне забезпечення для зняття та відправки даних. Перш за все це Arduino. Arduino - це електронний конструктор і зручна платформа швидкої розробки електронних пристроїв для новачків і професіоналів. Платформа користується величезною популярністю в усьому світі завдяки зручності і простоті мови програмування, а також відкритої архітектури і програмного коду. Пристрій програмується через USB без використання программаторов.

Arduino дозволяє комп'ютеру вийти за рамки віртуального світу в фізичний і взаємодіяти з ним. Пристрої на базі Arduino можуть отримувати інформацію про навколишнє середовище за допомогою різних датчиків, а також можуть управляти різними виконавчими пристроями. Мікроконтролер на платі програмується за допомогою мови Arduino (заснований на мові Wiring) і середовища розробки Arduino (заснована на середовищі Processing). Проекти пристроїв, засновані на Arduino, можуть працювати самостійно, або ж взаємодіяти з програмним забезпеченням на комп'ютері (напр .: Flash, Processing, MaxMSP). У класичній лінійці пристроїв Arduino в основному застосовуються мікроконтролери Atmel AVR.

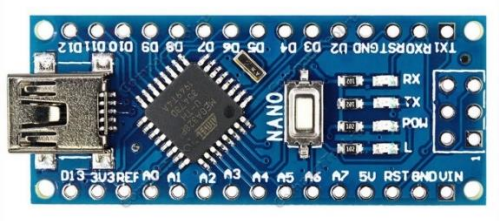

Рисунок 3.1 – Зовнішній вигляд мікроконтролера Arduino

Далі для більш професійного використання та розробки програм більш низького рівня та контролю подходить платформа STM32. STM32 - це платформа, в основі якої лежать мікроконтролери STMicroelectronics на базі ARM процесора, різні модулі та периферія, а також програмні рішення (IDE) для роботи з залізом. Рішення на базі stm активно використовуються завдяки продуктивності мікроконтролера, його вдалою архітектурі, малому енергоспоживанні, невеликій ціні. В даний час STM32 складається вже з декількох лінійок для самих різних призначень. Це популярна і дуже затребувана платформа, що дозволяє створювати професійні рішення для автоматизації в самих різних областях. На відміну від доступного Arduino, STM32 вимагає більш глибокого занурення в деталі, вона складніше для початківців. Це сімейство мікроконтролерів, що засноване на 32-бітних ядрах ARM Cortex-M7F, Cortex-M4F, Cortex-M3, Cortex-M0 + або Cortex-M0 зі скороченим набором інструкцій. STMicroelectronics (ST) має ліцензію на IP-процесори ARM від ARM Holdings. Дизайн ядра ARM має безліч параметрів, опцій, і ST вибирає індивідуальну конфігурацію для кожного мікроконтролера, при цьому додаючи свої власні периферійні пристрої до ядра мікроконтролера перед перетворенням дизайну в напівпровідникову пластину. У наступній таблиці представлені основні серії мікроконтролерів сімейства STM32 [12].

Таблиця 3.1 – Огляд мікропроцесорів серій мікроконтролерів STM32

Серія STM32	Ядро ARM CPU
L5	Cortex-M33
F7, H7	Cortex-M7F
F4, F3, L4, J	Cortex-M4F
F2, F1, L1, W, J	Cortex-M3
L0, J	Cortex-M0+
F0, J	Cortex-M0

Виробник ділить всі серії мікроконтролерів STM32 на 4 платформи (групи)

Таблиця 3.2 – Платформи мікроконтролерів STM32

Платформа мікроконтролерів	Назви серій, що входять в групу
Високопродуктивні	F2, F4, F7, H7
Широкого застосування	F0, G0, F1, F3, G4
Наднизького споживання	L0, L1, L4, L4+, L5
Бездротові	WB, WL

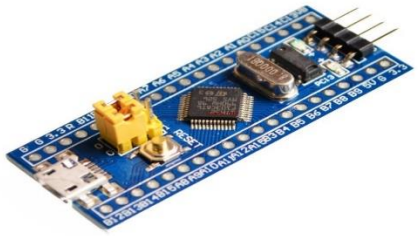

Рисунок 3.2 – Зовнішній вигляд мікроконтролера STM32

Розглядаючи сімейство контролерів STM32, також важливо згадати операційну систему Mbed OS, яка полегшує розробку під ARM.

Arm Mbed OS - це безкоштовна операційна система з відкритим кодом, розроблена спеціально для пристроїв в Інтернеті речей. Вона включає всі функції, необхідні для розробки підключеного продукту на основі мікроконтролера Arm Cortex-M, включаючи безпеку, підключення, RTOS (від англ. real-time operating system) та драйвери для датчиків та пристроїв вводу-виводу. Також вона представляє можливість роботи з декількома потомками, є необхідні інструменти для планування задач в операційній системі, що у перспективі використовує ресурси мікропроцесору ефективніше та підвищує цим енергоефективність [13].

Особливості ОС Mbed:

- Модульна (Необхідні бібліотеки автоматично включаються у пристрій, що дозволяє зосередитися на написанні коду програми.)
- Безпечна (Багатошарова безпека допомагає захистити рішення IoT (Internet of Things), від ізольованих доменів безпеки до Mbed TLS для безпечного зв'язку.)

- Підтримує багато протоколів зв’язку (Пропонується широкий спектр варіантів зв'язку з драйверами для Bluetooth Low Energy, 6LoWPAN, Mobile IoT (LPWA), Ethernet та WiFi.)

Також є ще одне, дуже вдале рішення. Для прототипу було обрано саме його. А саме одноплатний комп’ютер Raspberry Pi. Raspberry Pi - одноплатний комп'ютер, тобто різні частини комп'ютера, які зазвичай розташовуються на окремих платах, тут представлені на одній. До того ж ця плата має відносно невеликий розмір - приблизно 8,5 * 5,5 см. Це розмір приблизно банківської картки. Розробляється британською компанією Raspberry Pi Foundation. Цей одноплантий комп’ютер придбав широку популярність завдяки нескінченному діапазону можливостей. Минуло вже 8 років з моменту виходу першої моделі Raspberry Pi, і що тільки не було зроблено з цього маленького одноплатного комп'ютера - ігрові консолі, музичні плеєри, автомобільні навігатори та багато іншого.

Рисунок 3.3 – Зовнішній вигляд Raspberry Pi 3B

Взагалі існує багато різних моделей цих одноплатних комп’ютерів.

Таблиця 3.3 – Порівняння різних моделей Raspberry Pi

Модель	Рік випуску	Процесор	Об’єм оперативної пам’яті	Кількість ядер та тактова частота	Стандарт Wi-Fi	Швидкість Ethernet Мб/с	Кількість пінів
Pi 1 B	2012	ARM1176JZF-S	512 МБ	1x700 МГц	-	100	26
Pi 1 A	2013	ARM1176JZF-S	256 МБ	1x700	-	-	26

				МГц			
Pi 1 A+	2014	ARM1176JZF-S	256 МБ	1x700 МГц	-	-	40
Pi 1 B+	2014	ARM1176JZF-S	512 МБ	1x700 МГц	-	100	40
Pi 2 B	2015	ARM Cortex-A7	1 ГБ	4x900 МГц	-	100	40
Pi Zero	2015	ARM1176JZF-S	512 МБ	1x1 ГГц	-	0	40
Pi 3 B	2016	Cortex-A53 (ARM v8)	1 ГБ	4x1.2 ГГц	802.11n 4.1	100	40
Pi Zero W	2017	ARM1176JZF-S	512 МБ	1x1 ГГц	802.11n 4.0	0	40
Pi 3 A+	2018	Cortex-A53 (ARM v8)	512 МБ	4x1.4 ГГц	802.11ac 4.2	0	40
Pi 3 B+	2018	Cortex-A53 (ARM v8)	1 ГБ	4x1.4 ГГц	802.11ac 4.2	1000	40
Pi 4 B	2019	Cortex-A72 (ARM v8)	2, 4, 8 ГБ	4x1.5 ГГц	802.11ac 5.0	1000	40

З даного вибору було прийнято рішення використовувати саме модель Raspberry Pi Zero W v1.3. Для даної системи вона підходить найкраще тому, що має найменший розмір зі всіх цих моделей. В цій моделі використовується такий же процесор, що й у перший моделях, але з підвищеною тактовою частотою. Це досягається за допомогою «розгону» процесору. Розгін комп'ютерів - процес збільшення тактової частоти компонента комп'ютера понад штатних режимів з метою збільшення швидкості його роботи. Підвищення частоти може досягати максимального значення, при якому зберігається стабільність роботи системи в необхідному для користувача режимі. Слід враховувати, що при збільшенні частоти збільшується і тепловиділення. Тому при такому збільшенню частоти треба процесор, або інший компонент комп’ютера, забезпечити необхідною системою охолодження. Частіше всього такий підхід використовують самі користувачі, якщо вони хочуть підвищити продуктивність роботи процесору, але дана конфігурація йде з заводу та встановлена самим виробником. Також

ця плата має достатній об’єм оперативної пам’яті. А саме головне те, що на борту цієї плати знаходиться Wi-Fi модуль. Таким чином можна виділити такі переваги саме цієї моделі перед іншими для конкретної системи:

- Невеликий розмір
- Енергоефективний та достатньо швидкий процесор
- Достатня кількість оперативної пам’яті
- Вбудований модуль бездротового з’єднання
- Кількість пінів
- Підтримка сучасних версій операційних систем

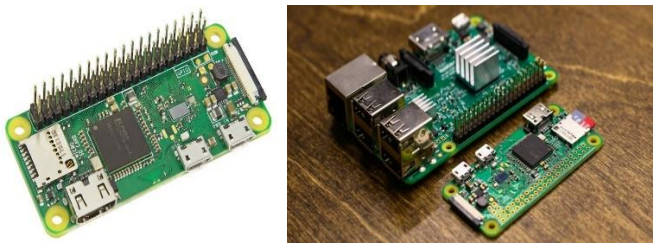

Рисунок 3.4 – Зовнішній вигляд та порівняння розмірів Raspberry Pi Zero W

Так як процесор міні-комп’ютеру, що представлений вище, має архітектуру ARM, то є багато операційних систем, які можна встановити на нього.

Найбільш розповсюджена і широко відома операційна система для міні-комп’ютера Raspberry Pi це Rasbperri Pi OS (раніше відома як Raspbian). Ця операційна система заснована на Debian. Існує декілька версій Raspbian, в тому числі Raspbian Stretch і Raspbian Jessie. Raspbian була створена Пітером Гріном і Майком Томпсоном. Ця операційна система представляється в якості незалежного проекту. З 2015 року Raspbian офіційно представлена Raspberry Pi Foundation в якості основної операційної системи для одноплатних комп'ютерів Raspberry Pi. Первісна збірка була виконана в червні 2012 року. Операційна система знаходиться в стадії активної розробки. Raspbian оптимізована для процесорів ARM, які використовуються

в лінійці комп'ютерів Raspberry Pi. В якості головного середовища робочого столу використовується PIXEL (Pi Improved Xwindows Environment, Lightweight). Ця система складається з модифікованої середовища робочого столу LXDE і менеджера вікон Openbox з новою темою і декількома іншими змінами. Є три збірки цієї операційної системи:

- Raspberry Pi OS з робочим столом та рекомендованим програмним забезпеченням – 2523МБ
- Raspberry Pi OS з робочим столом – 1127 МБ
- Raspberry Pi OS полегшена версія – 432 МБ

Своєї постійної пам’яті Raspberry Pi не має. У якості такої пам’яті використовується microSD карта пам’яті. Secure Digital Memory Card (SD) - формат карт пам'яті (флеш-пам'ять), розроблений SD Association (SDA) для використання в портативних пристроях. Тому образ операційної системи і встановлюється перед тим, як підключати цю карту до міні-комп’ютера і вводу у експлуатацію. Існують декілька популярних програм функціонал яких пропонує запис образу операційної системи на носій. А саме:

- Rufus
- Win32DiskImager
- Raspberry Pi Imager

Для встановлення образу системи було обрано програму Win32DiskImager. Операційна система, яка була обрана, це Raspberry Pi OS з робочим столом та рекомендованим програмним забезпеченням.

Після встановлення образу операційної системи йде її налаштування. А саме є два пункти, які треба зробити до того, як підключати карту пам’яті до міні-комп’ютера і робити перший запуск. Це налаштування підключення до бездротової точки доступу Wi-Fi, та включення служби SSH.

Підключення до без дротової точки доступу відбувається створенням файлу конфігурації з назвою wpa_supplicant.conf. Раніше в одноплатних комп’ютерах Raspberry Pi це потрібно було налаштовувати вручну. Насьогодні це можна робити до першого запуску. Після створення і налаштування цього файлу він поміщається у розділ /boot. У разі включення система сама знайде його і скопіює ці данні до потрібного файлу.

Також, як було сказано вище, можна і активувати можливість підключенню по SSH при першому запуску.

SSH - це комерційний продукт і надається на платній основі. Присутній і безкоштовна версія – OpenSSH. Більшість систем використовують саме OpenSSH версію. В силу свого відкритого вихідного коду, вона більш безпечна і зручна у використанні [14].

Переваги SSH протоколу:

- Дозволяє працювати віддалено на комп'ютері через командну оболонку.
- Шифрування здійснюється за допомогою різних алгоритмів.
- SSH дозволяє безпечно передавати практично будь-який мережевий протокол, це дає можливість передавати по шифрованому каналу звукові та відео файли.
- Стискає файли для їх подальшого шифрування і передачі.
- Захищає передачу даних по каналу і практично запобігає будь-яку можливість включення в встановлену сесію і перехопити дані.

Для автономної настройки SSH можна включити, помістивши файл з ім'ям ssh без розширення в завантажувальний розділ SD-карти. Коли Pi завантажується, він шукає файл ssh; якщо він знайдений, SSH включається, а потім файл видаляється. Вміст файлу не має значення: воно може містити або текст, або взагалі нічого.

Для підключення до Raspberry Pi по SSH треба використовувати SSH клієнти. Найпоширеніший клієнт для такого підключення це Putty. Putty - це безкоштовна програма для підключення до сервера через безпечне з'єднання SSH, Telnet, TCP або rlogin. Тобто, це тільки своєрідна оболонка, що відповідає за відображення: робота виконується на стороні віддаленого вузла. Грубо кажучи, вона застосовується для передачі команд сервера. Відбувається це приблизно за такою схемою: йде підключення до сервера за допомогою налаштованої Putty, вводиться в рядок команда, сервер її виконує. Дана утиліта налаштовує шрифти, кольори і дозвіл консолі, веде логи, зберігає у своїй пам'яті ключі авторизації, а також підтримує повноцінну роботу через проксі-сервер. При цьому, є абсолютно безкоштовною.

Але насьогодні можна і не встановлювати цей додаток. Засоби операційної системи Windows мають вбудований SSH клієнт. Для цього ми відкриваємо Windows Terminal і вводимо дані для входу. А саме адрес нашої плати, ім’я користувача та пароль. Знайти адрес Raspberry Pi можна за допомогою аналізаторів мережі або у панелі адміністратору маршрутизатору. Коли ми перший раз запускаємо нашу плату, то встановлюється стандартний логін і пароль, а саме логін pi, пароль raspberry. Після того, як відомі ці данні, вводиться дана команда для підключення, вводиться пароль и ми отримуємо доступ до нашого міні-комп’ютера.

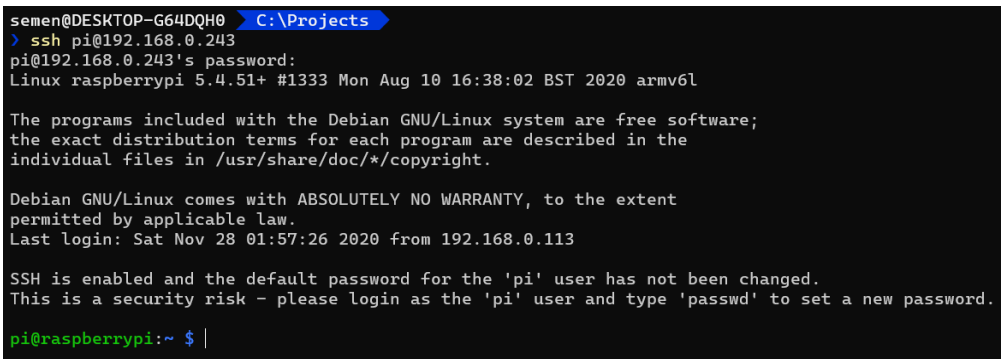

Рисунок 3.5 – Процес підключення до плати

Підключення датчика MH-Z19B до мікро-комп’ютера Raspberry Pi Zero W відбувається за рахунок підключенням до відповідних пінів.

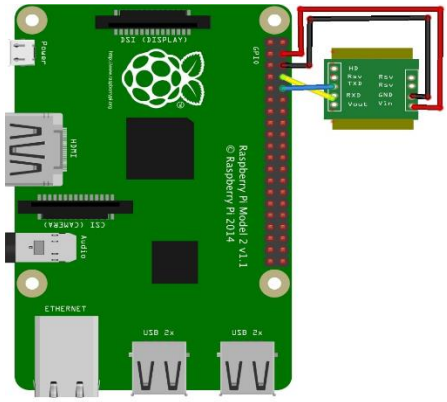

Рисунок 3.6 – Схема підключення датчика MH-Z19B до одногоплатного комп’ютера Raspberry Pi

Основний код для роботи з датчиком знаходиться у файлі DataSender.py. Щоб ця програма автоматично запускалась при кожному включенні одноплатного комп’ютера були здійснені деякі додаткові операції. Був створений спеціальний Bash скрипт launcher.sh. Цей скрипт запускає DataSender.py, який і виконує основну функцію отримання і передачі даних. Код файлу launcher.sh виглядає наступним чином:

```
#!/bin/sh
# launcher.sh

cd /
cd home/pi
sudo python DataSender.py
cd /
```

Код файлу DataSender.py приведений нижче:

```
import mh_z19
import time
import requests

while True:
    try:
        data = mh_z19.read()
        requests.packages.urllib3.disable_warnings()
```

```
requests.post('https://192.168.0.113:5000/api/values',verify=Fal
se, json={"SensorId": 1, "CO2": data['co2']})
        print("[" + time.strftime("%H:%M:%S", time.localtime())
+ "] CO2: " + str(data['co2']))
        time.sleep(5)
    except:
        print("Error of data reading or sending")
```

У цій програмі використовується три бібліотеки, які використовуються для отримання і відправки даних, а саме:

- time
- mh-z19b
- requests

time – модуль для роботи з часом у python. В даній програмі він використовується для створення затримки в п’ять секунд. Без цієї затримки дані відправлялись би дуже швидко і це спричинило би безглузде навантаження як на серверну частину, так і на саму плату. Обрано було затримку саме такого розміру тому, що сам датчик зчитує дані саме з інтервалом у п’ять секунд, тому не має сенсу отримувати дані з датчику, якщо вони не оновлені.

mh-z19b – це модуль для роботи з однойменним датчиком з одноплатними комп’ютерами Rasbperry Pi за допомогою мови python і способом підключення через UART. Відмінності інтерфейсу між кожною моделлю Raspberry Pi вирішені всередині цього модуля Тобто модуль mh-z19 автоматично визначає модель та зчитує з відповідного пристрою.

requests – модуль для мови програмування python і є стандартним інструментом для складання HTTP-запитів. Простий і акуратний API (з англ. Application Programming Interface) значно полегшує трудомісткий процес створення запитів. Таким чином, можна зосередитися на взаємодії зі службами і використанні даних в додатку.

Опис коду виглядає наступним чином. Командами `import mh_z19`, `import time`, `import requests` ми додаємо ці бібліотеки до нашої програми.

За допомогою `data = mh_z19.read()` ми отримуємо дані з датчика. Команда тяля вимикає попередження про безпеку пов'язаних з HTTPS. Так як сервер цієї системи використовує HTTPS протокол для більшого захисту і шифрування даних, що передаються, а не HTTP, то йде перевірка сертифікату. На комп'ютері, де проводилось тестування програмного продукту був встановлений сертифікат для розробки. І відключення цих попереджень значно зменшує об'єм логів, які зберігаються. Даною командою `requests.post('https://192.168.0.113:5000/api/values',verify=False, json={"SensorId": 1, "CO2": data['co2']})` ми відправляємо отримані дані з датчика на сервер за допомогою POST запиту, формуючи спеціальний json об'єкт.

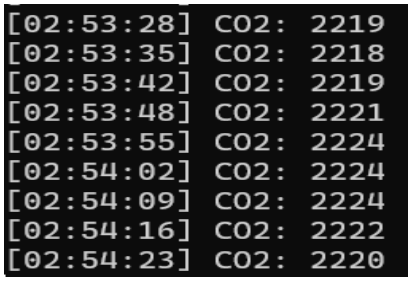

Рисунок 3.7 – Вивід скрипту DataSender.py у консоль

3.2 Розробка програми диспетчеризації

Розробка програми диспетчеризації процес при якому було використані сучасні технологій і бібліотеки для досягнення максимальної ефективності роботи і збільшення енергоефективності всієї системи.

Загальна структура проекту має даний вигляд. Детальніше про кожен компонент програмного продукту буде розписано нижче.

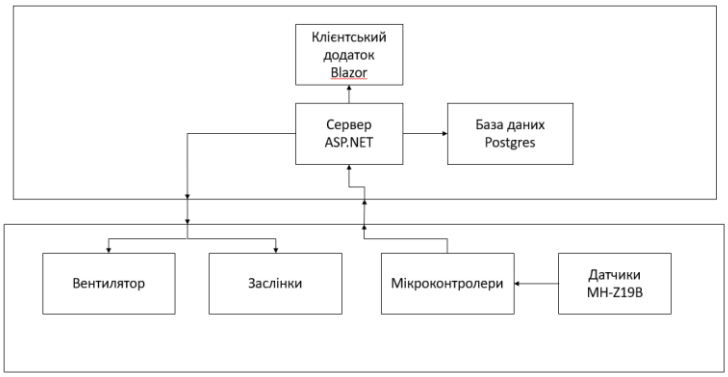

Рисунок 3.8 – Загальна структура системи

Створення програми диспетчеризації відбувалась на мові програмування C# та платформі .NET.

.NET - це безкоштовна крос-платформна платформа для розробників з відкритим кодом для створення багатьох різних типів додатків [15]. Можна виділити наступні її основні риси:

- Підтримка декількох мов
- Підтримка багатьох платформ
- Потужна бібліотека класів
- Різноманітність технологій
- Швидкість роботи

Сама серверна частина написана з використанням фреймворку ASP.NET, а саме ASP.NET Core. Платформа ASP.NET Core представляє технологію від компанії Microsoft, призначену для створення різного роду веб-додатків: від невеликих веб-сайтів до великих веб-порталів і веб-сервісів. ASP.NET Core може працювати поверх крос-платформної середовища .NET Core, яке може бути розгорнуте на основних популярних операційних системах: Windows, Mac OS, Linux. І таким чином, за допомогою ASP.NET Core з'являється можливість створювати крос-платформні додатки. Для розгортання веб-додатки можна використовувати традиційний IIS, або крос-платформний веб-сервер Kestrel [16]. Завдяки модульності фреймворка всі необхідні

компоненти веб-додатки можуть завантажуватися як окремі модулі через пакетний менеджер Nuget. Крім того, на відміну від попередніх версій платформи немає необхідності використовувати бібліотеку System.Web.dll.

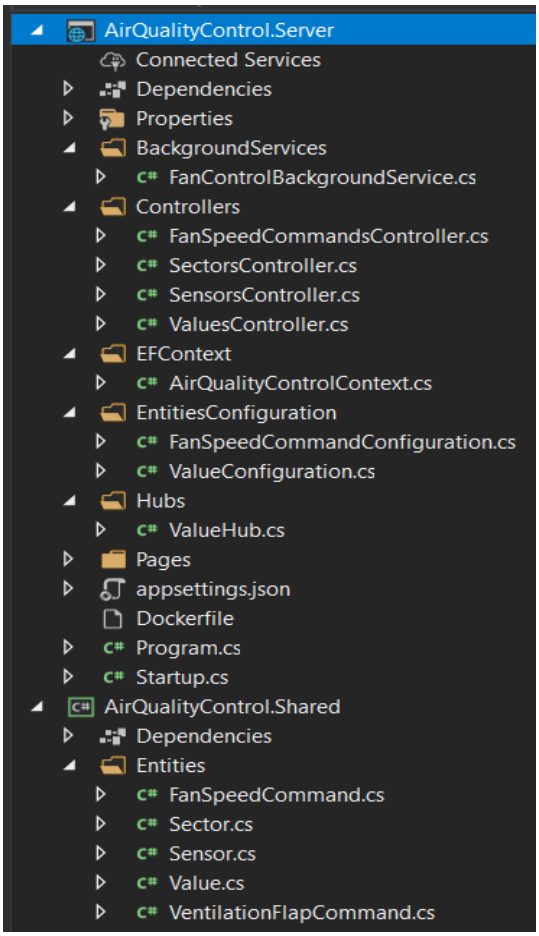

Рисунок 3.9 – Структура проекту серверної частини програмного продукту

У даному проекті використовувалась бібліотека Entity Framework Core для роботи з базою даних. Entity Framework Core (EF Core) являє собою об'єктно-орієнтовану, легковажну і доповнювану технологію від компанії Microsoft для доступу до даних. EF Core є ORM-інструментом (с англ. object-relational mapping - відображення даних на реальні об'єкти). Тобто EF Core дозволяє працювати базами даних, але надає більш високий рівень абстракції: EF Core дозволяє абстрагуватися від самої бази даних і її таблиць і працювати з даними незалежно від типу сховища. Якщо на фізичному рівні ми оперуємо таблицями, індексами, первинними і зовнішніми ключами, то на

концептуальному рівні, який нам пропонує Entity Framework, ми вже працюємо з об'єктами [17].

У папці Entities знаходяться ці самі сутності, які ми використовуємо для зберігання, обробки і відправлення інформації на клієнт. Було вибрано підхід Code First і тому бібліотека Entity Framework Core сама створює таблиці на основі цих сутностей.

Код файлу Sensor.cs

```
    public class Sector
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public int LowerThreshold { get; set; }
        public int UpperThreshold { get; set; }
        public List<Sensor> Sensors { get; set; }
        public List<FanSpeedCommand> FanSpeedCommands { get;
set; }
    }
```

За допомогою структури цього класу EF Core створює дану таблицю

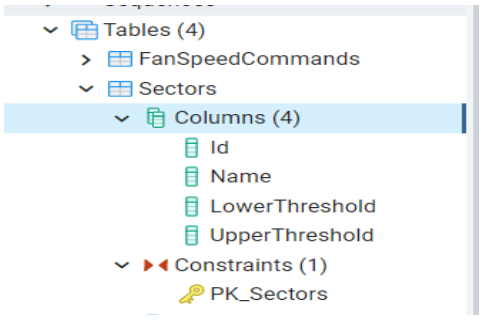

Рисунок 3.10 – Схема таблиці Sectors

Інші сутності використовуються аналогічно окрім VentilationFlapCommand. Ця сутність використовується тільки для передачі даних від серверу до самої заслінки.

Код файлу VentilationFlapCommand.cs

```
    public class VentilationFlapCommand
    {
        public int SensorId { get; set; }
        public int FlapLevel { get; set; }
    }
```

Як видно при передачі використовуються два поля: SensorId та FlapLevel, що означає ідентифікаційний номер сенсора до якого прив'язана заслінка і рівень її відкритості у процентах.

Основним класом конфігурації Entity Framework являється AirQualityControlContext. Саме у ньому ми вказуємо які сутності ми додаємо до контексту, які конфігурації ми приймаємо і які стартові дані буде мати наша база даних.

Код AirQualityControlContext.cs:

```
    public class AirQualityControlContext : DbContext
    {
        public DbSet<Sector> Sectors { get; set; }
        public DbSet<Sensor> Sensors { get; set; }
        public DbSet<Value> Values { get; set; }
        public  DbSet<FanSpeedCommand>  FanSpeedCommands  {  get;
set; }

        public
AirQualityControlContext(DbContextOptions<AirQualityControlConte
xt> options)
            : base(options)
        {
            Database.EnsureCreated();
        }

        protected   override   void   OnModelCreating(ModelBuilder
modelBuilder)
        {
            modelBuilder.ApplyConfiguration(new
ValueConfiguration());
            modelBuilder.ApplyConfiguration(new
FanSpeedCommandConfiguration());

            modelBuilder.Entity<Sector>().HasData(new Sector[]
            {
                new Sector
                {
                    Id = 1,
                    Name = "Home",
                    LowerThreshold = 1000,
                    UpperThreshold = 2000
                },

                new Sector
                {
                    Id = 2,
                    Name = "KPI",
```

```
                LowerThreshold = 800,
                UpperThreshold = 1000
            }
        });

        modelBuilder.Entity<Sensor>().HasData(new Sensor[]
        {
            new Sensor
            {
                Id = 1,
                SectorId = 1
            },
            new Sensor
            {
                Id = 2,
                SectorId = 2
            },
            new Sensor
            {
                Id = 3,
                SectorId = 2
            },
            new Sensor
            {
                Id = 4,
                SectorId = 2
            },
            new Sensor
            {
                Id = 5,
                SectorId = 2
            }
        });

        base.OnModelCreating(modelBuilder);
    }
}
```

У першій частині класу ми вказуємо сутності, які будуть додані. Наприклад `public DbSet<Sector> Sectors { get; set; }`. Команда `Database.EnsureCreated()` означає, що при створенні контексту ми перевіряємо чи створена база даних, якщо ні, то запускається процес її створення. За допомогою `modelBuilder.ApplyConfiguration(new ValueConfiguration())` ми додаємо додаткові конфігурації до наших сутностей.

Код файлу конфігурації виглядає наступним чином. Для прикладу буде приведений файл ValueConfiguration.cs:

```
    public class ValueConfiguration : IEntityTypeConfiguration<Value>
    {
        public void Configure(EntityTypeBuilder<Value> builder)
        {
            builder.HasKey(p => p.DateTime);
        }
    }
```

Так як, дана сутність не має поля з назвою Id, то Entity Framework не знає яке поле призначити як Primary Key. Тому за допомогою строчки `builder.HasKey(p => p.DateTime)` ми явно вказуємо яке поле буде нашим головним ключем.

Для отримання і відправки даних використовуються контролери. Всі контролери знаходяться у папці Controllers. На прикладі класу ValuesController буде описано основну задачу цих класів. Код ValuesController.cs:

```
    [Route("api/[controller]")]
    [ApiController]
    public class ValuesController : ControllerBase
    {
        private readonly ILogger<ValuesController> logger;
        private readonly AirQualityControlContext airQualityControlContext;
        private readonly IHubContext<ValueHub> hubContext;

        public ValuesController(ILogger<ValuesController> logger, AirQualityControlContext airQualityControlContext, IHubContext<ValueHub> hubContext)
        {
            this.logger = logger;
            this.airQualityControlContext = airQualityControlContext;
            this.hubContext = hubContext;
        }

        [HttpPost]
        public async Task<IActionResult> AddValue(Value value)
        {
            logger.LogInformation($"SensorId: {value.SensorId} CO2: {value.CO2}");
```

```
            value.DateTime = DateTime.UtcNow;
            airQualityControlContext.Values.Add(value);

            await airQualityControlContext.SaveChangesAsync();

            await
hubContext.Clients.All.SendAsync("ReceiveValue", value);

            return Ok();
        }

        [HttpGet]
        public IActionResult GetValues()
        {
            return Ok(airQualityControlContext.Values);
        }
    }
```

При створенні цього класу за допомогою Dependency Injection ми отримуємо об’єкти для роботи з логуваням, контекст для роботи за базою даних і контекст для роботи з бібліотекою SignalR. SignalR Core представляє бібліотеку від компанії Microsoft, яка призначена для створення додатків, що працюють в режимі реального часу. Зокрема, її можна використовувати разом з ASP.NET Core. SignalR використовує двонаправлений зв'язок для обміну повідомленнями між клієнтом і сервером, завдяки чому сервер може відправляти в режимі реального часу всім клієнтам деякі дані. Даний клас має два методи, а саме для отримання даних `public IActionResult GetValues()` і додавання даних `public async Task<IActionResult> AddValue(Value value)`. Розглядаючи код методу `GetValues()` ми можем спостерігати, що завдяки бібліотеці Entity Framework отримання значень відбувається за допомогою строки `return Ok(airQualityControlContext.Values)`, де ми, крім отримання, одразу і відправляємо відповідь. Метод додавання працює наступним чином. Спочатку за допомогою строки `logger.LogInformation($"SensorId: {value.SensorId} CO2: {value.CO2}")` логується отримана інформація від датчика. Це зроблена для більш швидкого знаходження поломки або

несправності. Тут `value.DateTime = DateTime.UtcNow` ми вказуємо поточний час на момент зберігання даних у стандарті UTC. За допомогою цих двох строк `await airQualityControlContext.SaveChangesAsync()` `await hubContext. Clients.All.SendAsync("ReceiveValue", value)` ми додаємо і зберігаємо нашу сутність. Ключове слово await означає, що метод буде виконуватися асинхронно. Це зроблено для того, щоб під час зберігання потік не блокувався, а продовжував свою роботу.

Для керування заслінками і швидкістю вентилятора використовується такий клас як FanControlBackgroundService. Він реалізований за допомогою такої можливості ASP.NET Core, як фонові процеси. У ASP.NET Core фонові завдання можуть бути реалізовані як розміщені служби. Розміщений сервіс - це клас із фоновою логікою завдань, що реалізує інтерфейс IHostedService. BackgroundService - базовий клас для реалізації тривалого IHostedService. Для запуску фонової служби викликається ExecuteAsync (CancellationToken). Цей сервіс ми підключаємо у файлі startup.cs за допомогою строки `services.AddHostedService<FanControlBackgroundService>()`, що надає можливість запустити логіку цього класу у фоні.

Код FanControlBackgroundService.cs:

```
    public class FanControlBackgroundService : BackgroundService
    {
        private readonly ILogger<FanControlBackgroundService>
logger;
        private readonly IHubContext<ValueHub> hubContext;
        private readonly IServiceScopeFactory scopeFactory;

        public
FanControlBackgroundService(ILogger<FanControlBackgroundService>
logger, IServiceScopeFactory scopeFactory, IHubContext<ValueHub>
hubContext)
        {
            this.logger = logger;
            this.scopeFactory = scopeFactory;
            this.hubContext = hubContext;
        }

        protected override async Task
ExecuteAsync(CancellationToken stoppingToken)
```

```
        {
            while (!stoppingToken.IsCancellationRequested)
            {
                using (var scope = scopeFactory.CreateScope())
                {
                    var dbContext =
scope.ServiceProvider.GetRequiredService<AirQualityControlContex
t>();

                    foreach (var sector in
dbContext.Sectors.Include(p => p.Sensors).ThenInclude(p =>
p.Values).ToList())
                    {
                        if (sector.Sensors is null ||
sector.Sensors.Count == 0)
                            continue;

                        var coefficient = (sector.UpperThreshold
- sector.LowerThreshold) / 100;

                        var fanSpeed = 0;

                        foreach (var sensor in sector.Sensors)
                        {
                            if (sensor.Values is null ||
sensor.Values.Count == 0)
                                continue;

                            var co2 =
sensor.Values.OrderByDescending(p => p.DateTime).First().CO2;
                            int flapLevel = (co2 -
sector.LowerThreshold) / coefficient;

                            if (flapLevel > 100)
                                flapLevel = 100;
                            if (flapLevel < 0)
                                flapLevel = 0;

                            fanSpeed += flapLevel;
                            await
hubContext.Clients.All.SendAsync("ReceiveFlapLevel", new
VentilationFlapCommand { SensorId = sensor.Id, FlapLevel =
flapLevel });
                            logger.LogInformation($"{sensor.Id}
{flapLevel}");
                        }

                        fanSpeed /= sector.Sensors.Count;

                        var fanSpeedCommand = new
FanSpeedCommand { SectorId = sector.Id, Speed = fanSpeed,
DateTime = DateTime.UtcNow };
```

```
logger.LogInformation($"{fanSpeedCommand.SectorId}
{fanSpeedCommand.Speed} {fanSpeedCommand.DateTime}");

                        await
hubContext.Clients.All.SendAsync("ReceiveFanSpeed",
fanSpeedCommand);

                        await
dbContext.FanSpeedCommands.AddAsync(fanSpeedCommand);
                    }

                    await dbContext.SaveChangesAsync();

                }

                await Task.Delay(3000, stoppingToken);
            }
        }
    }
```

Цей клас працює за наступним алгоритмом. Спочатку отримує всі значення кожного датчика сектору. Розраховує коефіцієнт `var coefficient = (sector.UpperThreshold - sector.LowerThreshold) / 100`, який буде враховуватися при розрахунках, а саме наскільки сильно треба відкрити заслінку. На даній стрічці `var co2 = sensor.Values.OrderByDescending(p => p.DateTime).First().CO2` ми отримуємо актуальне значення концентрації вуглекислого газу. Далі `int flapLevel = (co2 - sector.LowerThreshold) / coefficient` вираховуємо на скільки відсотків нам треба відкрити заслінку. Після додавання всіх цих значень `fanSpeed += flapLevel` і при діленні на кількість заслінок `fanSpeed /= sector.Sensors.Count` ми можемо знайти на який об’єм повітря ми можемо встановити вентилятор, щоб досягти балансу у раціональному використанні ресурсів. Далі ми ці дані відправляємо до клієнтської частини і зберігаємо в базу даних.

В якості бази даних використовується Postgres. PostgreSQL - це об'єктно-реляційна система управління базами даних (ОРСУБД, ORDBMS),

заснована на POSTGRES, Version 4.2 - програмою, розробленою на факультеті комп'ютерних наук Каліфорнійського університету в Берклі. У POSTGRES з'явилося безліч нововведень, які були реалізовані в деяких комерційних СУБД набагато пізніше.

Клієнтська частина написана за допомогою фреймворку Blazor. Blazor представляє UI-фреймворк для створення інтерактивних додатків, які можуть працювати як на стороні сервера, так і на стороні клієнта, на платформі .NET. У своєму розвитку фреймворк Blazor зазнав значного впливу сучасних фреймворків для створення клієнтських додатків - Angular, React, VueJS. Зокрема, це проявляється в ролі компонентів при побудові користувальницького інтерфейсу. У той же час і на стороні клієнта, і на стороні сервера при визначенні коду в якості мови програмування застосовується C #, замість JavaScript. А для опису візуального інтерфейсу використовуються стандартні HTML і CSS [18].

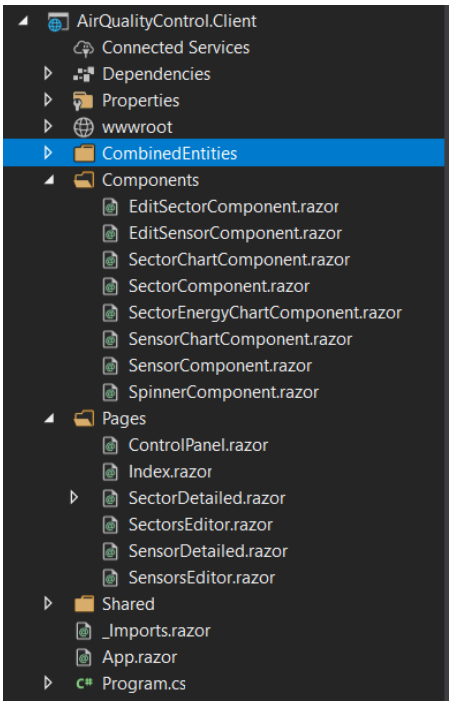

Рисунок 3.11 – Структура проекту клієнтського додатку

Так як Blazor це фреймворк, де основною неподільною частиною виступає компонент, то у проекті знаходиться група компонентів, які можуть використовуватися в інших компонентах. Ми можемо спостерігати папку Pages, але навіть сторінки виступають теж як компоненти. Сам компонент ділиться на дві частини. Перша частина це частина яка відповідає за відображення даного компоненту, а друга частина, частина коду, відповідає за його функціонал.

Наприклад візьмемо сторінку контрольної панелі. Там використовується компонент сектору.

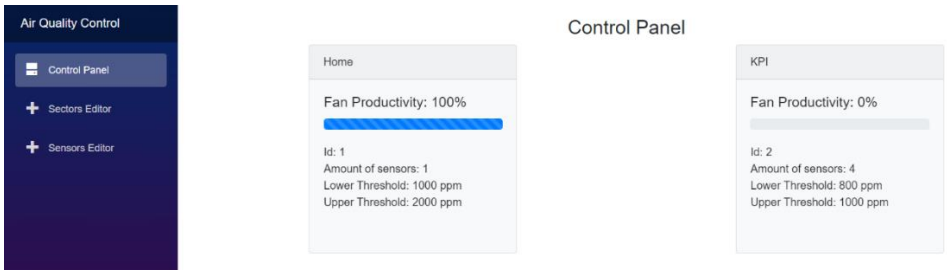

Рисунок 3.12 – Сторінка контрольної панелі клієнтського додатку

Схематично вона буде виглядати так.

Рисунок 3.13 – Сторінка контрольної панелі клієнтського додатку зображена схематично

У коді це виглядає наступним чином. Частина коду ControlPanel.razor:

```
if (SectorWithCurrentFanSpeeds.Count == 0)
{
    <SpinnerComponent />
}
else
```

```
{
    <div style="display: flex; flex-direction: row; justify-
content: flex-start; flex-wrap: wrap;">
        @foreach (var sectorWithCurrentFanSpeed in
SectorWithCurrentFanSpeeds)
        {
            <div style="margin: auto">
                <Animate Animation="Animations.ZoomIn"
Duration="TimeSpan.FromSeconds(0.5)"
Delay="TimeSpan.FromMilliseconds(animationDelay+= 100)">
                    <SectorComponent
Sector="sectorWithCurrentFanSpeed.Sector"
CurrentFanSpeed="sectorWithCurrentFanSpeed.CurrentFanSpeed" />
                </Animate>
            </div>
        }
    </div>
}
```

Де стрічка `<SectorComponent Sector="sectorWithCurrentFanSpeed.Sector"CurrentFanSpeed="sectorWithCurrentFanSpeed.CurrentFanSped"/>` як раз і виконується для створення компоненту сектору.

На сторінці сектору ми можем спостерігати кількість датчиків, що відносяться до цього сектору. Їх статус, поточне значення і значення у відсотках на скільки відкрита заслінка, яка відноситься до цього датчку.

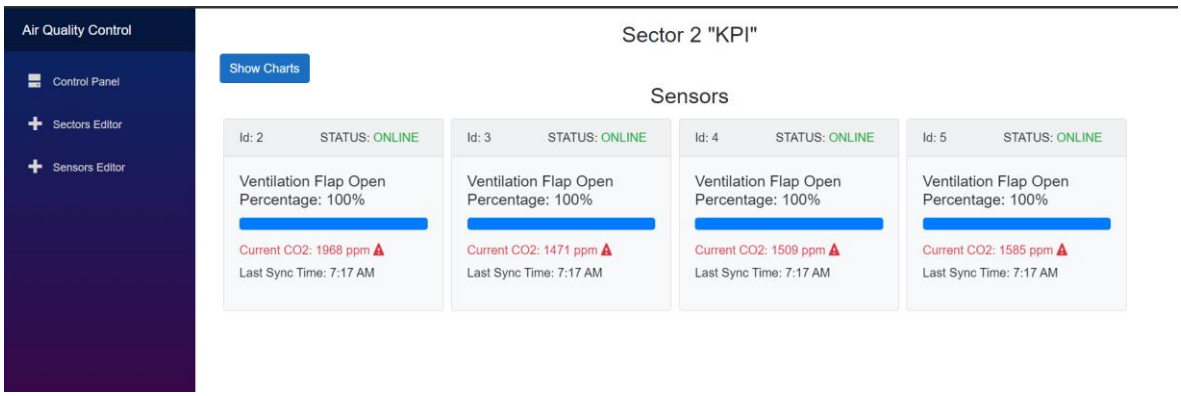

Рисунок 3.14 – Вигляд сторінки сектору

На сторінці датчика ми можемо спостерігати графік зміни концентрації вуглекислого газу у повітрі.

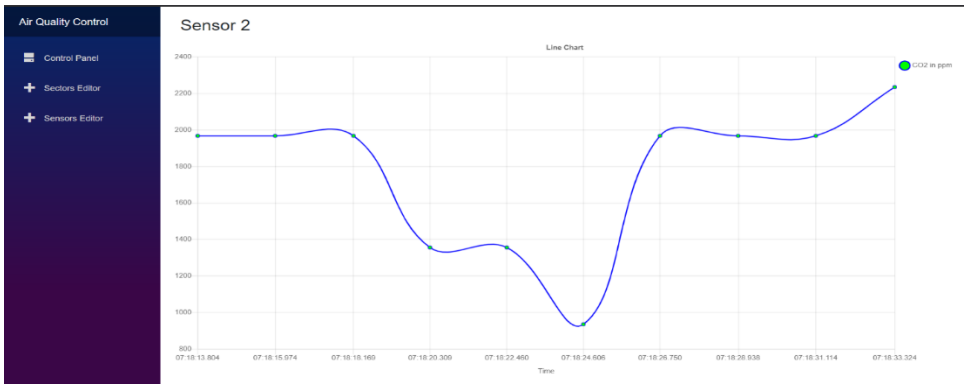

Рисунок 3.15 – Вигляд сторінки датчика

На сторінці редактору секторів ми можемо додавати, змінювати і видаляти сектори без необхідності зупинки всієї системи.

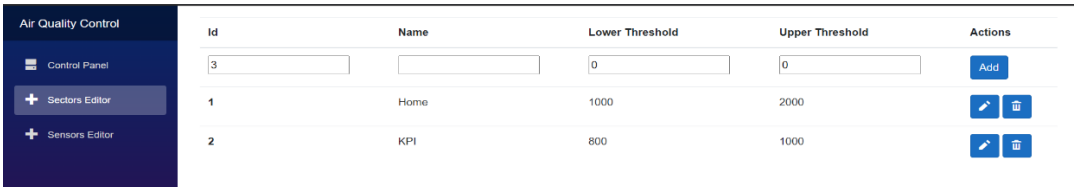

Рисунок 3.16 – Вигляд сторінки редагування секторів.

На сторінці редактору датчиків ми можемо додавати, змінювати і видаляти датчики, а також змінювати номер сектору, до якого вони прив’язані, також без необхідності зупинки всієї системи.

Рисунок 3.17 – Вигляд сторінки редагування датчиків.

Для запуску всіє системи використовується платформа для контейнеризації Docker. Код файлу docker-compose.yml:

```
version: '3.4'

services:
  airqualitycontrol.server:
    container_name: backend
    image: ${DOCKER_REGISTRY-}airqualitycontrolserver
    build:
      context: .
      dockerfile: AirQualityControl/Server/Dockerfile
    ports:
      - 5000:443

  postgresdb:
    container_name: postgresdb
    image: postgres
    restart: always
    environment:
      POSTGRES_PASSWORD: air
    ports:
      - 5432:5432
```

Це надає можливість запускати прогармний продукт у окремих конейнерах і незалежно від середовища. Тому, що контейнер має все необхідне для запуску програмного продукту.

3.3 Висновки

В даному розділі було розроблене програмне забезпечення контролерів для одноплатного комп’ютера Raspberry Pi Zero W для зчитування вимірювань з датчику вуглекислого газу MH-Z19B та відправку даних на сервер. Реалізовано серверну частину з можливістю зберігання даних, обробки та розрахунку значень для елементів вентиляції. Також розроблена клієнтська частина програмного забезпечення для відображення актуальних даних.

4 РОЗРОБКА СТАРТАП ПРОЕКТУ «AIR QUALITY CONTROL»

4.1 Опис ідеї проекту

В даному розділі розглянуто та описано ідею проекту. На сьогодні Інтелектуальні будівлі це майбутнє будівельних галузей. Зменшення витрат досягається за рахунок зменшення споживання енергії. Використання автоматичних систем вентиляції призводить до досягання даної цілі, а саме підвищенню її енергоефективності. Основні напрямки застосування, зміст ідеї та вигоди для користувача наведені в таблиці 4.1. Також з таблиці можна зрозуміти основні переваги та актуальність системи стартапу.

Таблиця 4.1 – Опис ідеї стартап проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Система контролю якості повітря та управління вентиляцією для енергоефективної вентиляції інтелектуальних будівель	Вимірювання концентрації вуглекислого газу у повітрі	Можливість відслідковувати кількість вуглекислого газу у повітрі, щоб у разі перевищення вживати необхідні заходи.
	Контроль вентиляції в залежності від отриманих даних	Підвищення енергоефективності за рахунок раціонального використання ресурсів
	Моніторинг та диспетчеризація системи	Відображення і обробка інформації для своєчасного виявлення поломок

Така система має бути реалізована та побудована таким чином щоб мати можливість гнучкого налаштування, високої стабільності та зручного користування. Даний продукт має актуальність на ринку та дотримується трендів сучасної розробки, що робить таку систему комерційно перспективною в сфері надання послуг. Для цього і пропонується розробка стартап проекту, який має гнучкість у налаштуванні, можливість до високої стабільності і ефективності, шо призводить до підвищення

енергоефективності, де при цьому з перспективою у майбутнє використовуються нові технології.

Виходячи з цього одним з розділів магістерської роботи є розробка стартап проекту. В цьому розділі розглядаються рішення для реалізації, актуального на сьогоднішній день проекту, який буде сильним гравцем на ринку і буде здатен створити конкуренції існуючим аналогам у перспективності, організованості, планом для розвитку ідеї та залучення інвесторів, фінансовому аналізі, аналізі ризиків і можливостей, маркетинговому плануванні.

Таблиця 4.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	Техніко-економічна характеристики ідеї	(потенційні) товари/концепції конкурентів				W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Vents	Tria-komm	Aclima			
1.	Вартість встановлення системи	7000	18000	9000	22000	-	-	+
2.	Ефективність алгоритму	Висока	Середня	Низька	Висока	-	-	+
3.	Гнучкість налаштування	Середня	Висока	Середня	Низька	-	+	-
4.	Розповсюдженість	Низька	Велика	Середня	Велика	+	-	-

Як результат цієї таблиці ми бачимо, що мій проект має дві сильні сторони, завдяки чому він спроможний скласти гідну конкуренції іншим рішенням.

4.2 Технологічний аудит проекту

Технологічний аудит - це спосіб перевірки технологічного стану підприємства (і галузі в цілому) за допомогою певних критеріїв, що дає

можливість виявлення її сильних і слабких сторін, що веде до формулювання стратегії, спрямованої на підвищення ефективності роботи підприємства (і галузі в цілому).

Методика технологічного аудиту є:

- алгоритм збору і обробки інформації;
- систему формування вихідних даних;
- алгоритм проведення аналізу та оцінки;
- оцінка ефективності існуючої і потенційно вдосконаленою систем;
- систематизація висновків;
- система ключових показників, що дозволяють виконувати постійний контроль ефективності роботи підприємства;
- рекомендації з оформлення технічної документації

Мета, з якою проводиться технічний аудит, полягає в зборі, аналізі, систематизації та комплексній оцінці придбаної під час перевірки інформації. Заходи необхідні, щоб виявити всі факти невідповідності на виробництві та, тим самим, попередити випадки травматизму, а також підвищити рівень безпеки умов праці для співробітників [19].

Під час проведення технічного аудиту перевіряються всі ліцензії, видані на присутнє обладнання та технології, що використовуються. Звіряються всі нормативні та розпорядчі документи, а також експлуатаційні журнали. Проводиться тестування обладнання і використовуваних контрольно-вимірювальних приладів. Знімаються всі необхідні свідчення з техніки, яка відповідає за облік і реєстрацію змін в роботі.

В даному розділі розглянуто технічні особливості, спектр можливих технічних рішень. Таблиця 5.3 використана для аналізу здійсненності проекту.

Таблиця 4.3 Технологічна здійсненність ідеї проекту

№ п/п	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій

1.	Демонстраційна система контролю якості повітря вентиляції	Недисперсійні інфрачервоні датчики вуглекислого газу	+	+
		Електрохімічні датчики вуглекислого газу	+	+
		Напівпровідникові датчики оксиду металу вуглекислого газу	-	+
		Передача даних через глобальну мережу	+	+
		Передача даних через локальну мережу	+	+
		Централізований сервер	+	-
		Сервер, який знаходиться в кожному приміщенні	+	+
Обрана технологія реалізації ідеї проекту: Система контролю якості повітря та управління вентиляцією для підвищення енергоефективності.				

Вентиляція за споживанням це існуючий підхід, який широко використовується в сучасному світі. Але мій проект пропонує новий підхід до розробки. Мій проект це централізована система, яка зачитує дані зі всіх датчиків і на їх основі формує яку саме потужність треба встановити на центральному вентиляторі. Також ця система включає в себе клієнтський додаток, який допомагає диспетчеру бачити завжди актуальну інформацію і у випадку поломки швидко її знайти. Саме з цією метою в даному проекті використовуються сучасні фреймворки для розробки програмного забезпечення, також спеціальний алгоритм для вирішення основної задачі і найбільш відповідний датчик. Використовується недисперсійний інфрачервоний датчики вуглекислого газу, так як він має найбільш приближений діапазон роботи до реальних потреб. Також використовується сервер, який знаходиться у кожному приміщенні і передача даних

відбувається через локальну мережу, тим самим ми можемо підвищити безпеку.

4.3 Аналіз ринкових можливостей запуску стартап-проекту

Виходу з стартапу на ринок передує його оцінювання як з точки зору можливого рівня прибутку, типу ринку, що визначає поведінку на ньому, так і з точки зору можливостей та загроз, які будуть супроводжувати стартап. Для оцінювання ринку використовується метод порівняння з конкурентами, що полягає в прогнозуванні власних показників через результати найбільш схожих конкурентів. Оскільки прямими конкурентами стартапів не часто бувають публічні компанії, які викладають свою звітність, то проблема даного методу в пошуку актуальної інформації. З оглядів, статей засновників проекту та результатів отриманих інвестицій, можна дізнатися їх оцінку обсягу ринку, прогнозований темп зростання і займану частку.

Рисунок 4.1 - Оцінювання потенційного ринку стартапу

Визначення ринкових можливостей та ризиків, які можна очікувати від ринку відіграють дуже важливу роль, оскільки дають можливість спроектувати та задати вектор розвитку проекту з урахуванням всіх нюансів ринкового положення, залученість клієнтів та пропозицій конкурентів. Таким

чином має сенс використання цих знань під час ринкового впровадження проекту(табл. 4.4).

Таблиця 4.4 Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	20
2	Загальний обсяг продаж, грн/ум.од	10000
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Відсутні
5	Специфічна вимоги до стандартизації та сертифікації	Дотримання норм та стандартів побудов вентиляції в будівлях
6	Середня норма рентабельності в галузі (або по ринку), %	30%

Розглянувши таблицю можна зрозуміти що даний ринок має хоч не велику кількість, але стабільних і великих гравців. Ринок цей має різко зростаючий характер. На ринку не так багато інноваційних компаній.

Далі визначені потенційні групи клієнтова та їх особливості. Також сформовано приблизний перелік вимог до системи вентиляції для кожної групи.

Таблиця 4.5 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до послуги
1	Система для автоматичного контролю вентиляції за потребою та диспетчеризації для інтелектуальних будівель	Сучасні офісні будівлі	Ефективність алгоритму та доступна ціна	Надійність, якість та ефективність
2		Великі приватні будинки	Прогресивність	Підвищення енергоефективності
3		Доповнення до існуючої інфраструктури, підприємства	Гнучке налаштування	Доступна ціна

Даний продукт добре покриває сферу організації енергоефективно вентиляції, таким чином збільшуючи шанси на те, що клієнт вибере саме мою систему. Ефективність алгоритму і взагалі системи за свою ціну надають дуже велику перевагу над конкурентами в цій самій сфері.

В таблиці 4.6 наведені фактори, які можуть в значній мірі перешкоджати подальшому розвитку стартап-проекту. Для того, щоб ці фактори не зашкоджували проекту робляться дії, які спрямовані на зміну ситуації на краще та налагодження процесу виробництва продукту. Дотримуючись цих дій з'являється можливість зберігати якість проекту.

Таблиця 4.6 Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1.	Крупні гравці	Недовіра клієнтів до нових менших гравців	Надавати якісний тат інноваційний продукт за доступну ціну
2.	Платформа	Платформа, на якій ведеться розробка може отримувати оновлення, які не включають зворотну сумісність	Вибір іншої платформи або створення компонентів на бібліотеках, які підтримуюсь максимальну кількість версій.
3.	Підвищення цін на комплектуючі	Ринок швидко змінюється, так же можуть і змінюватися ціни на комплектуючі залежно від попиту на них.	Додати підтримку більшості комплектуючих, щоб в майбутньому зменшити ризики на збитки та швидкий перехід на інші.
4.	Зміна стандартів для організації вентиляції	З часом все більше і більше екологія привертає увагу уряду і для цього він може змінювати стандарти.	Прогнозувати таки випадки і створення систем з розрахунком на те, що таке може статися.
5.	Архітектура	При створенні проекту настає момент, коли його підтримка більше неможлива або стає дуже дорогою.	При наявності таких факторів виконувати перехід на нову архітектуру та виділяти бюджет на покращення та підтримку програмного продукту.

Основними факторами загроз є крупні гравці, втрати зворотної сумісності платформи, підвищення цін на комплектуючі, зміна стандартів

для організації вентиляції та старіння архітектури.. Тому було представлено опис реакції компанії на такі фактори.

Також було створено таблицю факторів можливостей, що можуть сприяти ринковому впровадженню стартап-проекту. Такі фактори наведені в таблиці 4.7.

Таблиця 4.7. Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1.	Різке збільшення клієнтів	Одночасно багато підприємств захоче встановлення такої системи.	Масштабування підприємства і збільшення кількості співробітників.
2.	Потреба у віддаленому керуванні	Деякі клієнти попросять, щоб цією системою можна було керувати не тільки з будівлі де вона встановлена	Створення центрального серверу та оформлення передачі даних через глобальну мережу.
3.	Нові ідеї і пропозиції у клієнтів	У кожного клієнта можуть бути нові різні ситуація і для покращення продукту можуть з’являтися нові ідеї	Створення сервісу зворотного зв’язку та відділу підтримки клієнтів.
4.	Вихід нових перспективних технологів	Можливість з’явлення нових перспективних технологій і рішень.	Надання інженерам та іншим співробітникам доступ до освітніх матеріалів для самонавчання і покращення навичок.
5.	Клієнти з інших країн	Приплив нових клієнтів з інших країн.	Надання курсів іноземних мов та виділення бюджету для освоєння нових ринків.

Основні фактори можливостей це різке збільшення клієнтів, потреба у віддаленому керуванні, нові ідеї і пропозиції у клієнтів, вихід нових перспективних технологів та клієнти з інших країн. При виникненню ризиків також можливі і виникнення факторів можливостей. Якщо бути готовими до них, то це надає можливість нейтралізувати загрози розширенням або адаптації компанії.

Одним з важливих моментів появи на ринку є проведення ступеневого аналізу конкуренції на ринку. Також необхідно враховувати їх можливий вплив на ринок, що наведено в таблиці 4.8.

Таблиця 4.8. Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Досконала конкуренція	Є достатньо фірм і компаній, які надають такий вид послуг	Встановлення привабливої ціни та якості
2. Національний рівень	Локація і надання послуг конкурентів за даної країни і інших країн.	Швидкодія та якість
3. Внутрішньогалузева конкурентність	В основному використовується при проектуванні вентиляційної системи.	Надання умові, які повністю відповідають запитам клієнта
4.Товарно-видова конкурентність	Системи з однаковим принципом роботи	Створення нової, більш сучасної реалізації
5. Нецінова конкуренція	Різна якість продукції	Вдосконалення якості продукції та технології виробництва
6. Марочна конкуренція	Відомості марки конкурентів	Набір додаткової брендинг-команди

У даній сфері присутня велика конкуренція. Ринок налічує багато компаній з довготривалими контрактами, конкуренти стаж роботи яких більш як 10 років у цій сфері. Через великі можливості середнього прибутку, дуже жорстка боротьба за клієнта. В таблиці були наведені можливі дії компанії для поліпшення конкурентного становища.

Після аналізу конкуренції нижче наведений більш детальний аналіз умов конкуренції в галузі (табл.4.9).

Таблиця 4.9 Аналіз конкуренції в галузі за М. Портером

Складові аналіз	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники

у	Tria-komm	Vents	STMicroelectronics	Siemens	Місцеві вентиляції
Висновки:	Один з головних конкурентів.	Конкурент в майбутньому	Виробник мікросхем	Надає обладнання споживачам	Незалежні блоки вентиляцій, які встановлюють в окремі приміщення

Вказані компанії та фактори конкурентоспроможності дозволяють зрозуміти, що компанія має можливість протистояти на ринку. З таблиці видно можливості для конкуренто-спроможності стартапу.

В таблиці 4.10 нижче наведено обґрунтування факторів конкурентоспроможності.

Таблиця 4.10 Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Ефективність алгоритму	Використовується найефективніших алгоритм при якому враховуються дані всієї будівлі
2	Ціна	Майже найнижча ціна на ринку
3	Додаткові технології	Крім самої системи контролю ще додається програма диспетчеризації і відслідковування поломок
4	Зворотній зв’язок	Так як система зроблена за допомогою нових технологій, то є можливість додавати нові функції для різних клієнтів.
5	Якість	По якості конкуренти мають такі ж самі показники.

Таблиця 4.11. Порівняльний аналіз сильних та слабких сторін

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з власною компанією						
			–3	–2	–1	0	+1	+2	+3
1	Ефективність алгоритму	18		+					
2	Ціна	19			+				
3	Додаткові технології	17			+				
4	Час розробки	16						+	
5.	Підтримка	15				+			

Фінальним етапом ринкового аналізу можливостей впровадження проекту є складання SWOT-аналізу стартап-проекту на основі факторів загроз та факторів можливостей і також сильних і слабких сторін проекту.

SWOT - аналіз стартап-проекту представлений в таблиці 4.12.

Таблиця 4.12. – SWOT- аналіз стартап-проекту

Сильні сторони: 1. Ефективність алгоритму 2. Ціна 3 Додаткові технології 4 Підтримка	Слабкі сторони: 1.Час розробки
Можливості: Різке збільшення клієнтів Потреба у віддаленому керуванні Нові ідеї і пропозиції у клієнтів Вихід нових перспективних технологів Клієнти з інших країн	Загрози: Крупні гравці Платформа Підвищення цін на комплектуючі Зміна стандартів для організації вентиляції Архітектура

В даній таблиці були розроблені альтернативи ринкового впровадження стартап-проекту для просування його на ринок. А також було надано орієнтовний час їх реалізації.

Таблиця 4.13. Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Підвищення якості продукції та покращення алогритмів	Середня	6 місяці
2	Проведення рекламної компанії для підвищення впізнаваності компанії	Велика	6 місяців
3	Стратегія виходу ринок закордоном	-	3 роки

Отже, з зазначених альтернатив сильними сторонами стартапу є рекламної компанії для підвищення пізнаваності компанії.

4.4 Розроблення ринкової стратегії проекту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку, а саме опис цільових груп потенційних споживачів та інтенсивності конкуренції на ринку в обраному сегменті. Для роботи в обраних сегментах ринку було сформувано базову стратегію розвитку (табл. 4.14).

Таблиця 4.14. Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
	Компанії, які займаються побудовою бізнес-центрів або офісних будівель.	Так	Великий	Велика	Середній рівень входу
	Компанії сегменту побудови рішень для підприємств.	Так	Середній	Велика	Просто
	Компанії побудови великих приватних будинків	Так	Низький	Низька	Складно
Цільовими групами обрано: компанії, які займаються побудовою бізнес-центрів або офісних будівель та компанії сегменту побудови рішень для підприємств. Під час аналізу цільових груп потенційних споживачів було вирішено що проект буде працювати із компаніями, які займаються побудовою бізнес-центрів або офісних будівель.					

Базуючись на проведеному аналізі потенційних груп споживачів, було обрано такі цільові групи як компанії, які займаються побудовою бізнес-центрів або офісних будівель. Дані цільові групи мають найвищий попит на продукцію типу автоматичних систем вентиляції і відносно низький поріг входу на ринок.

Для роботи в обраному сегменті ринку необхідно сформувати базову стратегії розвитку стартап-проекту, охоплення ринку та ключові позиції конкурентоспроможності (табл.4.15).

Таблиця 4.15 – Базова стратегія розвитку

Обрана альтернатива розвитку проекту	Стратегія концентрованого зростання (Дана стратегія передбачає зміну ринку і (або) продукту. У разі дотримання стратегії компанія поліпшує програмний продукт або починає розробку нового, не змінюючи цільовий напрям. Щодо ринку, компанія шукає можливості покращення свого становища на нинішньому ринку або ж переходу на новий).
Стратегія охоплення ринку	Стратегія повного охоплення ринку (компанія надаватиме надійний продукт за доступні кошти з високою ефективністю алгоритму).
Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Покращення продукту враховуючі нові потреби клієнтів.
Базова стратегія розвитку	Стратегія диференціації (передбачає надання товару важливих з точки зору споживача відмітних властивостей, які роблять товар відмінним від товарів конкурентів. Така відмінність може базуватися на об'єктивних або суб'єктивних, відчутних і невідчутних властивостях товару(у ширшому розумінні – комплексі маркетингу), бути реальною або уявною. Інструментом реалізації стратегії диференціації є ринкове позиціонування.)

В таблиці вище зазначено обрані стратегії розвитку. У вигляді базової стратегії було обрано стратегію диференціації, з постійним покращенням продукту.

Наступною таблицею є вибір стратегії конкурентної поведінки (Таблиця 4.16).

Таблиця 4.16 – Вибір стратегії конкурентної поведінки

Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки

Ні	Так	Так (Так як всі системи вентиляції мають схожу структуру відмінність буде складати у оформленні нового підходу до її контролю)	Стратегія наслідування лідеру

Було прийнято використовувати стратегію наслідування лідерства тим самим роблячи схожий продукт, але з новим підходом або покращеннями враховуючи попередні помилки.

Далі з'являється можливість визначення стратегії позиціонування так як вже маємо визначену базову стратегію розвитку та характеристику потенційних клієнтів стартап-проекту.

Таблиця 4.17. Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
	Ефективність системи, доступна ціна та гнучкість.	Стратегія диференціації	Новий підхід до рішення існуючих рішень з кращим алгоритмом роботи	Ціна. Ефективність. Гнучкість.

Результатом даного підрозділу є формування ринкової позиції стартап-проекту за яким, визначається в якому напрямі буде працювати компанія на ринку.

4.5 Розробка маркетингової програми стартап-проекту

Розробка маркетингової програми стартап-проекти робиться для того, щоб про товар дізнались, товар почали і тим самим зростала би кількість клієнтів. створити ефективну маркетингову програму першим кроком є

формування маркетингової концепції товару, який отримає споживач. Для цього у таблиці 4.18 були визначені ключові переваги продукту.

Таблиця 4.18 – Визначення ключових переваг концепції потенційного товару

Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
Зменшення енергоспоживання вентиляціє у будівель.	Підвищення енергоефективності вентиляційної системи будівель.	Ефективний алгоритм та підхід керуванням системи вентиляції з програмним продуктом для диспетчеризації

З цієї таблиці можна зробити висновок, що для вирішення проблеми контролю вентиляції для підвищення енергоефективності розроблений спеціальний та новий алгоритм та також використана нова платформа і створений додатковий програмний продукт, а саме система диспетчеризації.

В цій таблиці була розглянута трирівнева маркетингова модель товару та використана для даного товару. Були зафіксовані такі пункти як ідея продукту, фізичні складові та особливості процесу його надання.

Таблиця 4.19. Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
I. Товар за задумом	Система контролю якості повітря та управління для енергоефективної вентиляції		
II. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	1. Алгоритм	Нм	Тх
	2. Ефективність	Нм	Тх
	3. Гнучкість	Нм	Тл
	4. Якість	Нм	Тх
	5. Доступна ціна	М	Е
	Якість: відповідає нормам ДСТУ 2388-94		
	Пакування: датчики вуглекислого газу, мікроконтролери, сервер, програмне забезпечення		
	Марка: “Air Quality Systems” зареєстрована ТМ. Назва товару “Air Quality Control”		
III. Товар із підкріпленням	До продажу: Підтримка з питань правильного встановлення		
	Після продажу: Оновлення програмного забезпечення		
Потенційний товар буде захищено від копіювання: патентування, не буде			

розкритий програмний код та алгоритм.

Як резльутат товар має п’ять технічних особливостей, чотири з яких є нематеріальними, а п’ятий матеріальним. Нормування якості товару проводиться згідно з ДСТУ 2388-94. До складу товару входять датчики вуглекислого газу, мікроконтролери, сервер, програмне забезпечення. Також на третьому рівні наведені переваги для споживача.

Наступним кроком є визначення меж встановлення ціни. Це передбачає аналіз цін товарів конкурентів, та доходів споживачів продукту (табл. 5.20).

Таблиця 4.20. Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1	50000 грн	30000 грн	130000 грн	8000/18000 грн
2	2700 грн	2300 грн	25000 грн	3000/5000 грн
3	20000 грн	20000 грн	300000 грн	10000/15000 грн

З огляду на аналіз ціна на товари-аналоги та товари-замінники були встановлені цінові межі. Як результат ми отримали межі, на які необхідно спиратись при встановленні ціни на товар.

Дана таблиця є наступним кроком для визначення оптимальної системи збуту. Вона включає в себе специфіку закупівельної поведінки цільових клієнтів, функції збуту, які має виконувати постачальник товару та оптимальну систему збуту.

Таблиця 4.21. Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Закупівля для встановлення в будівлю, яка будується або на стадії проектування	Доставка та консультація	0	Веб-сайт та таргетована реклама

2	Покращення вже існуючої системи	Доставка та консультація	0	Веб-сайт та таргетована реклама

Як результати ми маємо дві специфіки закупівельної поведінки цільових клієнтів, а саме закупівля для встановлення в будівлю, яка будується або на стадії проектування та вже покращення вже існуючої системи. Постачальник товару в двох випадках буде доставляти та консультувати клієнта.

Останнім пунктом розроблення складової маркетингової програми є розробка концепції маркетингових комунікацій. Вона що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів (табл. 22).

Таблиця 4.22. Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрано для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Клієнти, які шукають перспективні продукти, одне з яких встановлення енергоефективних вентиляцій	Чат-боти, соціальні мережі, офіційна електронна пошта, гаряча лінія, веб-сайт	Прийняття участі на технічних конференціях, реклама в інтернеті, розсилки по потенційним клієнтам	Представлення сильних сторін продукту, перспектив у ролі енергоефективності	Інфографіка сильних сторін та перспектив
2	Клієнти, які шукають перспективи для встановлення енергоефективних вентиляцій	Чат-боти, соціальні мережі, офіційна електронна пошта, гаряча лінія, веб-сайт	Прийняття участі на технічних конференціях, реклама в інтернеті, розсилки по потенційним клієнтам	Оформлення переваг, які буде мати клієнт, якщо вибере саме нашу продукцію, та прикріпити звіт	Інфографіка з перевагами та звітом

З огляду на специфіку поведінки цільових клієнтів та канали комунікації якими вони користуються ми можемо зробити висновок, що для залучення клієнтів, крім рекомендації інших клієнтів, є використання сучасних підходів до реклами. Правильно представлена інформація дасть змогу залучити багато нових клієнтів.

Бізнес-модель визначає місце стартапу в ланцюжку створення цінності (value chain). Бізнес-модель являє собою систему, що складається з таких компонентів бізнесу, як підприємництво, стратегія, економіка, фінанси, операції, конкурентні стратегії, маркетинг і стратегії розвитку компанії. Бізнес-модель є «скелетом» компанії. Усі процеси всередині компанії описуються її бізнес-моделлю і підпорядковуються їй. На початковому етапі життєдіяльності компанії необхідно запустити ці процеси і переконатись, що вони не конфліктують між собою, а ведуть компанію до отримання прибутку. Виникає потреба в деякому універсальному алгоритмі з перевірки бізнесмоделей, але для того, щоб бізнес-модель можна було швидко перевірити і описати, її необхідно шаблонізувати.

Існує багато видів бізнес-моделей. Бізнес-модель Canvas одна з найбільш зручних бізнес-моделей, яку доцільно використовувати розробникам стартапів. Бізнес-модель Canvas – один з інструментів стратегічного управління для підприємців, який дозволяє зробити опис запропонованого проекту або проаналізувати модель бізнесу, яка використовується, з позицій її ефективності та можливостей розвитку [20].

Даний вид бізнес-моделі має такі частини:

- споживчі сегменти – одна або декілька груп клієнтів, що охоплюється бізнес-моделлю. Групи клієнтів представляють різні сегменти, якщо: – відмінності в їх запитах обумовлюють відмінності в пропозиціях; – взаємодія здійснюється по різних каналах збуту;
- ціннісна пропозиція – сукупність переваг, які компанія готова запропонувати споживачеві. Наприклад, новизна, продуктивність,

виготовлення на замовлення, дизайн, бренд / статус, ціна, економія на витратах, зниження ризику, доступність, зручність / застосовність.

- канали збуту, що виконують ряд функцій, зокрема: підвищують ступінь поінформованості споживача про товари і послуги компанії; допомагають оцінити ціннісні пропозиції компанії; дозволяють споживачеві купувати певні товари та послуги; знайомлять споживача з ціннісними пропозиціями; забезпечують післяпродажне обслуговування [4; 5];
- взаємовідносини з клієнтами. Такі взаємовідносини передбачають персональна підтримка, самообслуговування, автоматизоване обслуговування, спільноти, спільне створення;
- потоки надходження доходу. До таких потоків включають продаж активів (товарів), плата за використання, оплата підписки, оренда / лізинг / рента, ліцензії, брокерські відсотки, реклама;
- ключові ресурси. Ключовими можуть бути і такі ресурси як: матеріальні, інтелектуальні, людські, фінансові;

Таблиця 4.23. Бізнес-модель проекту

Ключові партнери	Ключові види діяльності	Ціннісні пропозиції	Взаємодії з клієнтами	Споживчі сегменти
Фірми, що розробляють обладнання для вентиляційних систем Фінансові партнери Органи регулювання та уряд	Зберігання конкурентної ціни. Прийняття участі на технічних конференціях. Аналізувати відгуки Покращувати програмне	Ефективний алгоритм Програмне забезпечення для диспетчеризації та моніторингу Доступна ціна	Установка системи Обслуговування Зворотній зв’язок	Компанії, які займаються побудовою бізнес-центрів або офісних будівель. Компанії сегменту побудови рішень для підприємств.

<table>
<tr><td rowspan="3"></td><td>забезпечення</td><td rowspan="3"></td><td></td><td rowspan="3">Компанії побудови великих приватних будинків</td></tr>
<tr><td>Ключові ресурси

Розробники та інший персонал
Канали зв’язку для встановлення контакту з клієнтами
Зібрані дані</td><td>Канали збуту

Чат-боти, соціальні мережі, офіційна електронна пошта, гаряча лінія, веб-сайт</td></tr>
<tr><td colspan="2">Структура витрат

Реклама
Обслуговуючі платформи
Закупівля обладнання
Дослідження та розробка програмного забезпечення</td><td colspan="2">Потоки надходження доходу

Плата за встановлення та купівлю програмного продукту та компонентів, що потрібні для її роботи
Плата за модернізацію систем нашими компонентами</td></tr>
</table>

4.6 Висновки

Насьогодні технології починають використовуватися навіть в будівлях для контролю різних систем. Одна з таких систем це системи вентиляції. Збільшення використання ресурсів також веде до збільшення навантаження на інфраструктуру, мережу і екології. Тому один із способів зменшення цього фактору це підвищення енергоефективності. Для таких цілей з’явилась така галузь як розробка енергоефективної вентиляції, або по-іншому системи вентиляції за потребою.

Сьогодні існують достатньо компаній, які займаються розробкою продуктів такого виду. Кожна фірма може використовувати загальні принципи до досягнення цілі, або розробляти свої підходи. Актуальність

рішення, яке пропонує мій проект полягає у тому, що використовується спеціальний алгоритм, який враховує значення і покази датчиків зі всіх частин будівлі і знаходить оптимальний шлях як розподілити повітря по будівлі. З цього випливає те, що даний проект має можливість ринкової капіталізації.

Були виділені такі факти конкурентоспроможності як ефективність алгоритму, ціна, додаткові технології, зворотній зв'язок та якість.

Для стратегії розвитку стартапу було обрано стратегію диференціації. Данна стратегія передбачає надання товару важливих з точки зору споживача відмітних властивостей, які роблять товар відмінним від товарів конкурентів. Така відмінність може базуватися на об'єктивних або суб'єктивних, відчутних і невідчутних властивостях товару

Даний стартап-проект вважається перспективним тому, що він надає нові перспективи клієнтам в існуючій сфері, може скласти конкуренція та знайти свого клієнта. На кожні ризики та можливості було підібрану відповідну реакцію. Також проект пропонує додаткове програмне забезпечення для диспетчеризації і моніторингу якості повітря і в перспективі для вчасного виявлення поломок системи. Спостерігається тренд на інтелектуальні будівлі де використовується дана система. А так як даний програмний продукт гнучкий у налаштуванні, то він зможе задовольнити любого клієнта цієї сфери.

ВИСНОВКИ

Дана дисертація присвячена вирішенню актуальних задач пов’язаних з побудовою системи контролю якості повітря для енергоефективної вентиляції інтелектуальних будівель.

В ході виконання дисертації була розроблена структура системи контролю якості повітря для енергоефективної вентиляції та розроблено програмне забезпечення її роботи.

Для реалізації апаратної частини системи було використано датчик вуглекислого газу MH-Z19B. Для зчитування та передачі даних від датчика було використано одноплатний комп’ютер Raspberry Pi Zero W, також було розроблено апаратно-програмне для нього за допомогою мови програмування Python. Організовано локальне бездротове з’єднання функціональних елементів системи. Для передачі даних у системі було використано два протоколи: HTTP та WebSocket.

Було розроблено серверне програмне забезпечення на базі платформи .NET, фреймворку ASP.NET Core та реляційної бази даних PostgreSQL. Для розробки програмного забезпечення клієнтської частини системи було використано фреймворк Blazor та мову програмування C#.

Перевагами та особливостями даного програмного забезпечення являються гнучкість налаштування, надійність та здатність працювати з різною кількістю датчиків.

Розроблено стартап-проект, на основі якого було зроблено висновок, що дана система є перспективною та актуальною і може створити гідну конкуренцію на ринку.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Osama Omar. Intelligent building, definitions, factors and evaluation criteria of selection // Alexandria Engineering Journal, 2018 , pp.2903–2910.
2. В. Н. Ануфрієв Енергозбереження в будівлях // В. Н. Ануфрієв, Н. А. Андрієнко // Минск «Альтиора – Живые Краски», 2011, – 10 с.
3. Основні вимоги до систем вентиляції // Режим доступу: https://aniko-gas.ru/sistemy-otopleniya/sistema-i-kondicionirovaniya.html
4. Семенко Д. М. Автоматизація керування вентиляцією офісних приміщень // Семенко Д. М., Стаценко О. В. // Збірник праць XII науково-практичної конференції студентів, аспірантів, та молодих вчених “Погляд у майбутнє приладобудування” Київ – 2019, - 478 – 480 с.
5. Види систем вентиляції // Режим доступу: https://sovet-ingenera.com/vent/raschety/vidy-sistem-ventilyacii.html
6. Що показує датчик забруднення повітря? // Режим доступу: https://musorish.ru/chto-pokazyvaet-datchik-zagryazneniya-vozduha/
7. CO2 Sensors: Which Type Should You Be Looking For? // Режим доступу: https://learn.kaiterra.com/en/air-academy/carbon-dioxide-sensors
8. Конфігуровані контролери електроприводу серії IRMCK // Режим доступу: https://www.compel.ru/lib/55071
9. Siemens S7-200 Програмований контролер // Режим доступу: https://www.siemens-pro.ru/components/s7-200.htm
10. ADAPT Damper // Режим доступу: https://www1.swegon.com/Global/PDFs/Flow%20control/WISE%20gen.1/_ru/ADAPTD.pdf
11. Стаценко О. В. Розрахунок електромагнітних втрат енергії в регульованих асинхронних двигунах з короткозамкненим ротором // Стаценко О. В. // Вісник КНУТД. – 2013, – с. 158-166.

12. STM32 32-bit Arm Cortex MCUs // Режим доступу: https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html
13. Mbed OS // Режим доступу: https://os.mbed.com/mbed-os/
14. SSH. Навіщо і для чого? // Режим доступу: https://hyperhost.ua/info/ru/ssh-zachem-i-dlya-chego
15. Мова C # і платформа .NET Core // Режим доступу: https://metanit.com/sharp/
16. Керівництво по ASP.NET Core 3 // Режим доступу: https://metanit.com/sharp/aspnet5/
17. Що таке Entity Framework Core // Режим доступу: https://metanit.com/sharp/entity-frameworkcore/1.1.php
18. Blazor // Режим доступу: https://metanit.com/sharp/blazor/
19. Технологічний аудит // Режим доступу: https://techinservice.com.ua/tech-checking/teh-audit/
20. О. А. Гавриш Розробка стартап проектів // Гавриш, О. А., Бояринова К. О., Копішинська К. О. // Київ КПІ ім. Ігоря Сікорського 2019, – 68 с.

ДОДАТОК А. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

А.1 Програмне забезпечення для зчитування і відправку даних з датчику

А.1.1 Основний Python скрипт DataSender.py

```
import mh_z19
import time
import requests

while True:
    try:
        data = mh_z19.read()
        requests.packages.urllib3.disable_warnings()

requests.post('https://192.168.0.113:5000/api/values',verify=Fal
se, json={"SensorId": 1, "CO2": data['co2']})
        print("[" + time.strftime("%H:%M:%S", time.localtime())
+ "] CO2: " + str(data['co2']))
        time.sleep(5)
    except:
        print("Error of data reading or sending")
```

А.1.2 Скрипт для запуска Python скрипта при включенні системи launcher.sh

```
#!/bin/sh
#launcher.sh

sudo python3 /home/pi/humidity.py
```

А.2 Програмне забезпечення серверної частини системи

А.2.1 AirQualityControl.Server

А.2.1.1 Файл конфігурації appsettings.json

```
{

  "Logging": {

    "LogLevel": {

      "Default": "Information",

      "Microsoft": "Warning",

      "Microsoft.Hosting.Lifetime": "Information"

    }
```

```
  },

  "AllowedHosts": "*",

  "ConnectionStrings": {

    "DefaultConnection":
"Host=postgresdb;Port=5432;Database=AirQualityControl;Username=p
ostgres;Password=air"

  }

}
```

A.2.1.2 Файл основного класу Program.cs

```
public class Program

    {

        public static void Main(string[] args)

        {

            CreateHostBuilder(args).Build().Run();

        }


        public static IHostBuilder CreateHostBuilder(string[]
args) =>

            Host.CreateDefaultBuilder(args)

                .ConfigureWebHostDefaults(webBuilder =>

                {

                    webBuilder.UseStartup<Startup>();

                });

    }
```

A.2.1.2 Файл основних налаштувань Startup.cs

```
public class Startup

    {

        public Startup(IConfiguration configuration)
```

```
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this
method to add services to the container.
        // For more information on how to configure your
application, visit
https://go.microsoft.com/fwlink/?LinkID=398940
        public void ConfigureServices(IServiceCollection
services)
        {
            services.AddSignalR();
            services.AddControllersWithViews();
            services.AddRazorPages();
            services.AddResponseCompression(opts =>
            {
                opts.MimeTypes =
ResponseCompressionDefaults.MimeTypes.Concat(
                    new[] { "application/octet-stream" });
            });

services.AddDbContextPool<AirQualityControlContext>(options =>

options.UseNpgsql(Configuration.GetConnectionString("DefaultConn
ection")));

services.AddHostedService<FanControlBackgroundService>();
        }
```

```
        // This method gets called by the runtime. Use this
method to configure the HTTP request pipeline.

        public void Configure(IApplicationBuilder app,
IWebHostEnvironment env)

        {

            if (env.IsDevelopment())

            {

                app.UseDeveloperExceptionPage();

                app.UseWebAssemblyDebugging();

            }

            else

            {

                app.UseExceptionHandler("/Error");

                // The default HSTS value is 30 days. You may
want to change this for production scenarios, see
https://aka.ms/aspnetcore-hsts.

                app.UseHsts();

            }

            app.UseResponseCompression();

            app.UseHttpsRedirection();

            app.UseBlazorFrameworkFiles();

            app.UseStaticFiles();


            app.UseRouting();


            app.UseEndpoints(endpoints =>

            {

                endpoints.MapRazorPages();

                endpoints.MapControllers();

                endpoints.MapHub<ValueHub>("/valuehub");
```

```
                endpoints.MapFallbackToFile("index.html");

            });

        }

    }
```

A.2.1.3 Файл контейнера Dockerfile

```
FROM mcr.microsoft.com/dotnet/core/aspnet:3.1-buster-slim AS
base

WORKDIR /app

EXPOSE 80

EXPOSE 443


FROM mcr.microsoft.com/dotnet/core/sdk:3.1-buster AS build

WORKDIR /src

COPY
["AirQualityControl/Server/AirQualityControl.Server.csproj",
"AirQualityControl/Server/"]

COPY
["AirQualityControl/Shared/AirQualityControl.Shared.csproj",
"AirQualityControl/Shared/"]

COPY
["AirQualityControl/Client/AirQualityControl.Client.csproj",
"AirQualityControl/Client/"]

RUN dotnet restore
"AirQualityControl/Server/AirQualityControl.Server.csproj"

COPY . .

WORKDIR "/src/AirQualityControl/Server"

RUN dotnet build "AirQualityControl.Server.csproj" -c Release -o
/app/build


FROM build AS publish

RUN dotnet publish "AirQualityControl.Server.csproj" -c Release
-o /app/publish
```

```
FROM base AS final

WORKDIR /app

COPY --from=publish /app/publish .

ENTRYPOINT ["dotnet", "AirQualityControl.Server.dll"]
```

А.2.1.4 Файл фонового сервісу FanControlBackgroundService.cs

```
public class FanControlBackgroundService : BackgroundService

    {

        private readonly ILogger<FanControlBackgroundService>
logger;

        private readonly IHubContext<ValueHub> hubContext;

        private readonly IServiceScopeFactory scopeFactory;



        public
FanControlBackgroundService(ILogger<FanControlBackgroundService>
logger, IServiceScopeFactory scopeFactory, IHubContext<ValueHub>
hubContext)

        {

            this.logger = logger;

            this.scopeFactory = scopeFactory;

            this.hubContext = hubContext;

        }



        protected override async Task
ExecuteAsync(CancellationToken stoppingToken)

        {

            while (!stoppingToken.IsCancellationRequested)

            {

                using (var scope = scopeFactory.CreateScope())

                {
```

```
                    var dbContext =
scope.ServiceProvider.GetRequiredService<AirQualityControlContex
t>();



                    foreach (var sector in
dbContext.Sectors.Include(p => p.Sensors).ThenInclude(p =>
p.Values).ToList())

                    {

                        if (sector.Sensors is null ||
sector.Sensors.Count == 0)

                            continue;



                        var coefficient = (sector.UpperThreshold
- sector.LowerThreshold) / 100;



                        var fanSpeed = 0;



                        foreach (var sensor in sector.Sensors)

                        {

                            if (sensor.Values is null ||
sensor.Values.Count == 0)

                                continue;



                            var co2 =
sensor.Values.OrderByDescending(p => p.DateTime).First().CO2;

                            int flapLevel = (co2 -
sector.LowerThreshold) / coefficient;



                            if (flapLevel > 100)

                                flapLevel = 100;

                            if (flapLevel < 0)

                                flapLevel = 0;
```

```
                            fanSpeed += flapLevel;

                            await
hubContext.Clients.All.SendAsync("ReceiveFlapLevel", new
VentilationFlapCommand { SensorId = sensor.Id, FlapLevel =
flapLevel });

                            logger.LogInformation($"{sensor.Id}
{flapLevel}");

                        }


                        fanSpeed /= sector.Sensors.Count;


                        var fanSpeedCommand = new
FanSpeedCommand { SectorId = sector.Id, Speed = fanSpeed,
DateTime = DateTime.UtcNow };



logger.LogInformation($"{fanSpeedCommand.SectorId}
{fanSpeedCommand.Speed} {fanSpeedCommand.DateTime}");


                        await
hubContext.Clients.All.SendAsync("ReceiveFanSpeed",
fanSpeedCommand);


                        await
dbContext.FanSpeedCommands.AddAsync(fanSpeedCommand);

                    }


                    await dbContext.SaveChangesAsync();


                }


                await Task.Delay(3000, stoppingToken);
```

```
            }

        }

    }
```

A.2.1.5 Файл контролеру FanSpeedCommandsController.cs

```
[Route("api/[controller]")]

    [ApiController]

    public class FanSpeedCommandsController : ControllerBase

    {

        private readonly ILogger<FanSpeedCommandsController>
logger;

        private readonly AirQualityControlContext
airQualityControlContext;

        private readonly IHubContext<ValueHub> hubContext;



        public
FanSpeedCommandsController(ILogger<FanSpeedCommandsController>
logger, AirQualityControlContext airQualityControlContext,
IHubContext<ValueHub> hubContext)

        {

            this.logger = logger;

            this.airQualityControlContext =
airQualityControlContext;

            this.hubContext = hubContext;

        }

    }
```

A.2.1.6 Файл контролеру SectorsController.cs

```
[Route("api/[controller]")]

    [ApiController]

    public class SectorsController : ControllerBase

    {
```

```
        private readonly ILogger<SectorsController> logger;

        private readonly AirQualityControlContext
airQualityControlContext;

        private readonly IHubContext<ValueHub> hubContext;



        public SectorsController(ILogger<SectorsController>
logger, AirQualityControlContext airQualityControlContext,
IHubContext<ValueHub> hubContext)

        {

            this.logger = logger;

            this.airQualityControlContext =
airQualityControlContext;

            this.hubContext = hubContext;

        }



        [HttpGet]

        public async Task<IActionResult> GetAllSectors()

        {

            return Ok(await
airQualityControlContext.Sectors.OrderBy(p =>
p.Id).ToListAsync());

        }



        [HttpGet("{id}/sensorscount")]

        public async Task<IActionResult>
GetCountOfSensorsBySector(int id)

        {

            var sector = await
airQualityControlContext.Sectors.Include(p =>
p.Sensors).SingleOrDefaultAsync(p => p.Id == id);

            return Ok(sector.Sensors.Count);

        }
```

```
        [HttpGet("{id}")]
        public async Task<IActionResult> GetSectorById(int id)
        {
            var sector = await
airQualityControlContext.Sectors.SingleAsync(p => p.Id == id);
            return Ok(sector);
        }

        [HttpGet("{id}/sensors")]
        public IActionResult GetAllSensordBySector(int id)
        {
            var sensors =
airQualityControlContext.Sensors.Where(p => p.SectorId == id);
            return Ok(sensors);
        }

        [HttpGet("{id}/fanspeedcommands")]
        public IActionResult GetFanSpeedCommandsBySector(int id,
int limit)
        {
            var fanSpeedCommands =
airQualityControlContext.FanSpeedCommands.OrderByDescending(p =>
p.DateTime).Where(p => p.SectorId == id).Take(limit);

            return Ok(fanSpeedCommands);
        }

        [HttpPost]
        public async Task<IActionResult> AddSector(Sector
sector)
```

```
        {
            airQualityControlContext.Add(sector);
            await airQualityControlContext.SaveChangesAsync();
            return Ok();
        }

        [HttpPut]
        public async Task<IActionResult> EditSector(Sector
sector)
        {
            airQualityControlContext.Update(sector);
            await airQualityControlContext.SaveChangesAsync();

            return NoContent();
        }

        [HttpDelete("{id:int}")]
        public async Task<IActionResult> DeleteSector(int id)
        {
            var sector = new Sector { Id = id };
            airQualityControlContext.Attach(sector);
            airQualityControlContext.Remove(sector);
            await airQualityControlContext.SaveChangesAsync();

            return Ok();
        }
    }
```

A.2.1.7 Файл контролеру SensorsController.cs

```
[Route("api/[controller]")]

    [ApiController]

    public class SensorsController : ControllerBase

    {

        private readonly ILogger<SensorsController> logger;

        private readonly AirQualityControlContext
airQualityControlContext;

        private readonly IHubContext<ValueHub> hubContext;



        public SensorsController(ILogger<SensorsController>
logger, AirQualityControlContext airQualityControlContext,
IHubContext<ValueHub> hubContext)

        {

            this.logger = logger;

            this.airQualityControlContext =
airQualityControlContext;

            this.hubContext = hubContext;

        }



        [HttpGet("{id}/values")]

        public IActionResult GetValuesBySensor(int id, int
limit)

        {

            var values = airQualityControlContext.Values.Where(p
=> p.SensorId == id).OrderByDescending(p =>
p.DateTime).Take(limit);



            return Ok(values);

        }
```

```
        [HttpGet]
        public IActionResult GetAllSensors()
        {
            var sensors = airQualityControlContext.Sensors;

            return Ok(sensors);
        }

        [HttpPut]
        public async Task<IActionResult>EditSensor(Sensor
sensor)
        {
            airQualityControlContext.Update(sensor);
            await airQualityControlContext.SaveChangesAsync();

            return NoContent();
        }

        [HttpPost]
        public async Task<IActionResult> AddSensor(Sensor
sensor)
        {
            airQualityControlContext.Add(sensor);
            await airQualityControlContext.SaveChangesAsync();
            return Ok();
        }

        [HttpDelete("{id:int}")]
        public async Task<IActionResult> DeleteSensor(int id)
```

```
        {
            var sensor = new Sensor { Id = id };
            airQualityControlContext.Attach(sensor);
            airQualityControlContext.Remove(sensor);
            await airQualityControlContext.SaveChangesAsync();

            return Ok();
        }
    }
```

A.2.1.8 Файл контролеру ValuesController.cs

```
public class ValuesController : ControllerBase
    {
        private readonly ILogger<ValuesController> logger;
        private readonly AirQualityControlContext
airQualityControlContext;
        private readonly IHubContext<ValueHub> hubContext;

        public ValuesController(ILogger<ValuesController>
logger, AirQualityControlContext airQualityControlContext,
IHubContext<ValueHub> hubContext)
        {
            this.logger = logger;
            this.airQualityControlContext =
airQualityControlContext;
            this.hubContext = hubContext;
        }

        [HttpPost]
        public async Task<IActionResult> AddValue(Value value)
```

```
        {
            logger.LogInformation($"SensorId: {value.SensorId}
CO2: {value.CO2}");

            value.DateTime = DateTime.UtcNow;
            airQualityControlContext.Values.Add(value);

            await airQualityControlContext.SaveChangesAsync();

            await
hubContext.Clients.All.SendAsync("ReceiveValue", value);

            return Ok();
        }

        [HttpGet]
        public IActionResult GetValues()
        {
            return Ok(airQualityControlContext.Values);
        }
    }
```

A.2.1.9 Файл конфігурації Entity Framework AirQualityControlContext.cs

```
public class AirQualityControlContext : DbContext
    {
        public DbSet<Sector> Sectors { get; set; }
        public DbSet<Sensor> Sensors { get; set; }
        public DbSet<Value> Values { get; set; }
        public DbSet<FanSpeedCommand> FanSpeedCommands { get;
set; }
```

```
        public
AirQualityControlContext(DbContextOptions<AirQualityControlConte
xt> options)
            : base(options)
        {
            Database.EnsureCreated();
        }

        protected override void OnModelCreating(ModelBuilder
modelBuilder)
        {
            modelBuilder.ApplyConfiguration(new
ValueConfiguration());
            modelBuilder.ApplyConfiguration(new
FanSpeedCommandConfiguration());

            modelBuilder.Entity<Sector>().HasData(new Sector[]
            {
                new Sector
                {
                    Id = 1,
                    Name = "Home",
                    LowerThreshold = 1000,
                    UpperThreshold = 2000
                },

                new Sector
                {
                    Id = 2,
```

```
            Name = "KPI",
            LowerThreshold = 800,
            UpperThreshold = 1000
        }
    });

    modelBuilder.Entity<Sensor>().HasData(new Sensor[]
    {
        new Sensor
        {
            Id = 1,
            SectorId = 1
        },
        new Sensor
        {
            Id = 2,
            SectorId = 2
        },
        new Sensor
        {
            Id = 3,
            SectorId = 2
        },
        new Sensor
        {
            Id = 4,
            SectorId = 2
        },
```

```
                new Sensor
                {
                    Id = 5,
                    SectorId = 2
                }
            });

            base.OnModelCreating(modelBuilder);
        }
    }
```

A.2.1.9 Файл конфігурації сутності FanSpeedCommand FanSpeedCommandConfiguration.cs

```
    public class FanSpeedCommandConfiguration :
IEntityTypeConfiguration<FanSpeedCommand>
    {
        public void Configure(EntityTypeBuilder<FanSpeedCommand>
builder)
        {
            builder.HasKey(p => p.DateTime);
        }
    }
```

A.2.1.10 Файл конфігурації сутності Value ValueConfiguration.cs

```
public class ValueConfiguration :
IEntityTypeConfiguration<Value>
    {
        public void Configure(EntityTypeBuilder<Value> builder)
        {
```

```
            builder.HasKey(p => p.DateTime);
        }
    }
```

A.2.2 AirQualityControl.Shared

A.2.2.1 Файл сутності FanSpeedCommand FanSpeedCommand.cs

```
public class FanSpeedCommand
    {
        public DateTime DateTime { get; set; }
        public int Speed { get; set; }
        public int SectorId { get; set; }
        public Sector Sector { get; set; }
    }
```

A.2.2.2 Файл сутності Sector Sector.cs

```
public class Sector
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public int LowerThreshold { get; set; }
        public int UpperThreshold { get; set; }
        public List<Sensor> Sensors { get; set; }
        public List<FanSpeedCommand> FanSpeedCommands { get;
set; }
    }
```

A.2.2.3 Файл сутності Sensor Sensor.cs

```
public class Sensor
```

```
    {
        public int Id { get; set; }
        public int SectorId { get; set; }
        public Sector Sector { get; set; }
        public List<Value> Values { get; set; }
    }
```

A.2.2.4 Файл сутності Value Value.cs

```
public class Value
    {
        public DateTime DateTime { get; set; }
        public int SensorId { get; set; }
        public Sensor Sensor { get; set; }
        public int CO2 { get; set; }
    }
```

A.2.2.5 Файл сутності VentilationFlapCommand VentilationFlapCommand.cs

```
public class VentilationFlapCommand
    {
        public int SensorId { get; set; }
        public int FlapLevel { get; set; }
    }
```

A.2.3 Проект для контейнеризації docker-compose

A.2.2.5 Файл конфігурації docker-compose.yml

```
version: '3.4'
```

```
services:
  airqualitycontrol.server:
    container_name: backend
    image: ${DOCKER_REGISTRY-}airqualitycontrolserver
    build:
      context: .
      dockerfile: AirQualityControl/Server/Dockerfile
    ports:
      - 5000:443


  postgresdb:
    container_name: postgresdb
    image: postgres
    restart: always
    environment:
      POSTGRES_PASSWORD: air
    ports:
      - 5432:5432
```

А.3 Програмне забезпечення клієнтської частини системи

A.3.1 AirQualityControl.Client

A.3.1.1 Файл комбінованої сутності SensorWithCurrentValue.cs

```
public class SensorWithCurrentValue
    {
        public Sensor Sensor { get; set; }
        public Value Value { get; set; }
        public VentilationFlapCommand VentilationFlapCommand {
get; set; }
```

```
    }
```

A.3.1.2 Файл компоненту EditSectorComponent.razor

```
@using AirQualityControl.Shared.Entities

@using System.Net.Http

@inject HttpClient Http

@using AirQualityControl.Client.Pages


@if (isEditable)

{

    <tr>

        <th scope="row">@Sector.Id</th>

        <td><input @bind="@Sector.Name" /></td>

        <td><input type="number" @bind="@Sector.LowerThreshold"
/></td>

        <td><input type="number" @bind="@Sector.UpperThreshold"
/></td>

        <td>

            <button type="button" class="btn btn-primary"
@onclick="Save">Save</button>

        </td>

    </tr>

}

else

{

    <tr>

        <th scope="row">@Sector.Id</th>

        <td>@Sector.Name</td>

        <td>@Sector.LowerThreshold</td>
```

```
        <td>@Sector.UpperThreshold</td>

        <td>

            <button type="button" class="btn btn-primary"
@onclick="EnableEdit"><span class="oi oi-
pencil"></span></button>

            <button type="button" class="btn btn-primary"
@onclick="Delete"><span class="oi oi-trash"></span></button>

        </td>

    </tr>

}


@code {

    [Parameter]

    public Sector Sector { get; set; } = new Sector();


    [Parameter]

    public SectorsEditor SectorsEditor { get; set; }


    private bool isEditable = false;


    private async Task Save()

    {

        await Http.PutAsJsonAsync("api/sectors", Sector);

        isEditable = false;

    }


    private void EnableEdit()

    {

        isEditable = true;
```

```
    }


    private async Task Delete()

    {

        var result = await
Http.DeleteAsync($"api/sectors/{Sector.Id}");

        if(result.StatusCode == System.Net.HttpStatusCode.OK)

            SectorsEditor.DeleteSector(Sector);

    }

}
```

A.3.1.3 Файл компоненту EditSensorComponent.razor

```
@using AirQualityControl.Shared.Entities

@using System.Net.Http

@inject HttpClient Http

@using AirQualityControl.Client.Pages


@if (isEditable)

{

    <tr>

        <th scope="row">@Sensor.Id</th>

        <td style="width:800px">

            <select class="custom-select"
@bind="Sensor.SectorId">

                @foreach (var id in Sectors.Select(p => p.Id))

                {

                    <option value="@id">@id</option>

                }

            </select>
```

```
        </td>
        <td>
            <button type="button" class="btn btn-primary"
@onclick="Save">Save</button>
        </td>
    </tr>
}
else
{
    <tr>
        <th scope="row">@Sensor.Id</th>
        <td style="width: 800px">@Sensor.SectorId</td>
        <td>
            <button type="button" class="btn btn-primary"
@onclick="EnableEdit"><span class="oi oi-
pencil"></span></button>
            <button type="button" class="btn btn-primary"
@onclick="Delete"><span class="oi oi-trash"></span></button>
        </td>
    </tr>
}
@code {

    [Parameter]
    public Sensor Sensor { get; set; } = new Sensor();

    [Parameter]
    public List<Sector> Sectors { get; set; }

    [Parameter]
```

```
    public SensorsEditor SensorsEditor { get; set; }

    private bool isEditable = false;

    private async Task Save()
    {
        await Http.PutAsJsonAsync("api/sensors", Sensor);
        isEditable = false;
    }

    private void EnableEdit()
    {
        isEditable = true;
    }

    private async Task Delete()
    {
        var result = await
Http.DeleteAsync($"api/sensors/{Sensor.Id}");
        if (result.StatusCode == System.Net.HttpStatusCode.OK)
            SensorsEditor.DeleteSensor(Sensor);
    }
}
```

A.3.1.4 Файл компоненту SectorChartComponent.razor

```
@using ChartJs.Blazor.Charts
@using ChartJs.Blazor.ChartJS.Common.Properties
@using ChartJs.Blazor.ChartJS.Common.Enums
```

```
@using ChartJs.Blazor.ChartJS.Common.Axes

@using ChartJs.Blazor.ChartJS.Common.Axes.Ticks

@using ChartJs.Blazor.ChartJS.Common.Handlers

@using ChartJs.Blazor.ChartJS.Common.Time

@using ChartJs.Blazor.ChartJS.LineChart

@using ChartJs.Blazor.Util

@inject IJSRuntime JsRuntime

@using AirQualityControl.Shared.Entities

@using Microsoft.AspNetCore.SignalR.Client

@inject NavigationManager NavigationManager

@implements IDisposable

@using System.Net.Http

@inject HttpClient Http


<ChartJsLineChart @ref="_lineChartJs" Config="@_lineConfig"
Width="600" Height="300" />


@code

{

    [Parameter]

    public int SectorId { get; set; }


    LineConfig _lineConfig;

    ChartJsLineChart _lineChartJs;

    LineDataset<TimeTuple<int>> _tempDataSet;


    private HubConnection hubConnection;
```

```
public void Dispose()
{
    _ = hubConnection.DisposeAsync();
}

protected override async Task OnInitializedAsync()
{
    _lineConfig = new LineConfig
    {
        Options = new LineOptions
        {
            Animation = new Animation { Duration = 0 },
            Responsive = true,
            Title = new OptionsTitle
            {
                Display = true,
                Text = "Line Chart"
            },
            Legend = new Legend
            {
                Position = Position.Right,
                Labels = new LegendLabelConfiguration
                {
                    UsePointStyle = true
                }
            },
            Tooltips = new Tooltips
            {
```

```
                    Mode = InteractionMode.Nearest,
                    Intersect = false
                },
                Scales = new Scales
                {
                    xAxes = new List<CartesianAxis>
{
                        new TimeAxis
                        {
                            Distribution =
TimeDistribution.Linear,
                            Ticks = new TimeTicks
                            {
                                Source = TickSource.Data
                            },
                            Time = new TimeOptions
                            {
                                Unit =
TimeMeasurement.Millisecond,
                                Round =
TimeMeasurement.Millisecond,
                                TooltipFormat = "DD.MM.YYYY
HH:mm:ss:SSS",
                                DisplayFormats =
TimeDisplayFormats.DE_CH
                            },
                            ScaleLabel = new ScaleLabel
                            {
                                LabelString = "Time"
                            }
```

```
                    }
                }
            },
            Hover = new LineOptionsHover
            {
                Intersect = true,
                Mode = InteractionMode.Y
            }
        }
    };

    _tempDataSet = new LineDataset<TimeTuple<int>>
    {
        BackgroundColor = ColorUtil.ColorString(0, 255, 0,
1.0),
        BorderColor = ColorUtil.ColorString(0, 0, 255, 1.0),
        Label = "Fan Productivity in %",
        Fill = false,
        BorderWidth = 2,
        PointRadius = 3,
        PointBorderWidth = 1,
        SteppedLine = SteppedLine.False
    };

    _lineConfig.Data.Datasets.Add(_tempDataSet);

    var fanSpeedCommands = await
Http.GetFromJsonAsync<List<FanSpeedCommand>>($"api/sectors/{Sect
orId}/fanspeedcommands?limit=10");
```

```
        fanSpeedCommands.OrderBy(p =>
p.DateTime.ToLocalTime()).ToList().ForEach(p =>
_tempDataSet.Add(new TimeTuple<int>(new
Moment(p.DateTime.ToLocalTime()), p.Speed)));



        hubConnection = new HubConnectionBuilder()

        .WithUrl(NavigationManager.ToAbsoluteUri("/valuehub"))

        .Build();



        hubConnection.On<FanSpeedCommand>("ReceiveFanSpeed",
(value) =>

        {

            if (value.SectorId != SectorId)

                return;




            _tempDataSet.Add(new TimeTuple<int>(new
Moment(value.DateTime.ToLocalTime()), value.Speed));

            if (_tempDataSet.Data.Count > 10)

            {

                _tempDataSet.RemoveAt(0);

            }

            StateHasChanged();

        });



        await hubConnection.StartAsync();

    }

}
```

A.3.1.5 Файл компоненту SectorComponent.razor

```
@using AirQualityControl.Shared.Entities

@using System.Net.Http

@inject HttpClient Http

@using BlazorAnimate


<div class="card bg-light mb-3" style="max-width: 18rem; min-
height: 300px; min-width: 300px;">

    <div class="card-header">@Sector.Name</div>

    <div class="card-body">


        @if (CurrentFanSpeed == null)

        {

            if (sensorsCount > 0)

            {

                <Animate Animation="Animations.FadeIn"
Duration="TimeSpan.FromSeconds(0.5)"
Delay="TimeSpan.FromMilliseconds(0)">

                    <SpinnerComponent />

                </Animate>

            }

            else

            {

                <h4>No sensors avaliable</h4>

            }


        }

        else

        {
```

```
            <Animate Animation="Animations.FadeIn"
Duration="TimeSpan.FromSeconds(0.5)"
Delay="TimeSpan.FromMilliseconds(0)">

                <h5 class="card-title">Fan Productivity:
@CurrentFanSpeed%</h5>

                <div class="progress">

                    <div class="progress-bar progress-bar-
striped progress-bar-animated" role="progressbar" aria-
valuenow="75" aria-valuemin="0" aria-valuemax="100"
style="width: @progressBar"></div>

                </div>

            </Animate>

        }


        <p class="card-text" style="margin-top: 20px">

            Id: @Sector.Id <br />

            Amount of sensors:  @sensorsCount<br />

            Lower Threshold: @Sector.LowerThreshold ppm <br />

            Upper Threshold: @Sector.UpperThreshold ppm

        </p>

        <a href="/sectors/@Sector.Id" class="stretched-
link"></a>

    </div>

</div>


@code {

    [Parameter]

    public Sector Sector { get; set; }


    [Parameter]

    public int? CurrentFanSpeed { get; set; }
```

```
    string progressBar => $"{CurrentFanSpeed}%";


    int sensorsCount = 0;


    protected override async Task OnInitializedAsync()

    {

        sensorsCount = await
Http.GetFromJsonAsync<int>($"api/sectors/{Sector.Id}/sensorscoun
t");

    }

}
```

A.3.1.6 Файл компоненту SectorEnergyChartComponent.razor

```
@using ChartJs.Blazor.Charts

@using ChartJs.Blazor.ChartJS.Common.Properties

@using ChartJs.Blazor.ChartJS.Common.Enums

@using ChartJs.Blazor.ChartJS.Common.Axes

@using ChartJs.Blazor.ChartJS.Common.Axes.Ticks

@using ChartJs.Blazor.ChartJS.Common.Handlers

@using ChartJs.Blazor.ChartJS.Common.Time

@using ChartJs.Blazor.ChartJS.LineChart

@using ChartJs.Blazor.Util

@inject IJSRuntime JsRuntime

@using AirQualityControl.Shared.Entities

@using Microsoft.AspNetCore.SignalR.Client

@inject NavigationManager NavigationManager

@implements IDisposable

@using System.Net.Http
```

```
@inject HttpClient Http


<ChartJsLineChart @ref="_lineChartJs" Config="@_lineConfig"
Width="600" Height="300" />


@code {
    [Parameter]
    public int SectorId { get; set; }


    LineConfig _lineConfig;
    ChartJsLineChart _lineChartJs;
    LineDataset<TimeTuple<int>> _tempDataSet;


    private HubConnection hubConnection;


    public void Dispose()
    {
        _ = hubConnection.DisposeAsync();
    }


    protected override async Task OnInitializedAsync()
    {
        _lineConfig = new LineConfig
        {
            Options = new LineOptions
            {
                Animation = new Animation { Duration = 0 },
                Responsive = true,
```

```
                Title = new OptionsTitle
                {
                    Display = true,
                    Text = "Line Chart"
                },
                Legend = new Legend
                {
                    Position = Position.Right,
                    Labels = new LegendLabelConfiguration
                    {
                        UsePointStyle = true
                    }
                },
                Tooltips = new Tooltips
                {
                    Mode = InteractionMode.Nearest,
                    Intersect = false
                },
                Scales = new Scales
                {
                    xAxes = new List<CartesianAxis>
{
                        new TimeAxis
                        {
                            Distribution =
TimeDistribution.Linear,
                            Ticks = new TimeTicks
                            {
```

```
                                Source = TickSource.Data
                            },
                            Time = new TimeOptions
                            {
                                Unit =
TimeMeasurement.Millisecond,
                                Round =
TimeMeasurement.Millisecond,
                                TooltipFormat = "DD.MM.YYYY
HH:mm:ss:SSS",
                                DisplayFormats =
TimeDisplayFormats.DE_CH
                            },
                            ScaleLabel = new ScaleLabel
                            {
                                LabelString = "Time"
                            }
                        }
                    }
                },
                Hover = new LineOptionsHover
                {
                    Intersect = true,
                    Mode = InteractionMode.Y
                }
            }
        };

        _tempDataSet = new LineDataset<TimeTuple<int>>
        {
```

```
            BackgroundColor = ColorUtil.ColorString(0, 255, 0,
1.0),

            BorderColor = ColorUtil.ColorString(0, 0, 255, 1.0),

            Label = "Energy in %",

            Fill = false,

            BorderWidth = 2,

            PointRadius = 3,

            PointBorderWidth = 1,

            SteppedLine = SteppedLine.False

        };


        _lineConfig.Data.Datasets.Add(_tempDataSet);


        var fanSpeedCommands = await
Http.GetFromJsonAsync<List<FanSpeedCommand>>($"api/sectors/{Sect
orId}/fanspeedcommands?limit=10");

        fanSpeedCommands.OrderBy(p =>
p.DateTime.ToLocalTime()).ToList().ForEach(p =>
_tempDataSet.Add(new TimeTuple<int>(new
Moment(p.DateTime.ToLocalTime()),
Convert.ToInt32(System.Math.Pow(Convert.ToDouble(p.Speed) / 100,
3) * 100))));


        hubConnection = new HubConnectionBuilder()

        .WithUrl(NavigationManager.ToAbsoluteUri("/valuehub"))

        .Build();


        hubConnection.On<FanSpeedCommand>("ReceiveFanSpeed",
(value) =>

        {

            if (value.SectorId != SectorId)

                return;
```

```
            _tempDataSet.Add(new TimeTuple<int>(new
Moment(value.DateTime.ToLocalTime()),
Convert.ToInt32(System.Math.Pow(Convert.ToDouble(value.Speed) /
100, 3) * 100)));

            if (_tempDataSet.Data.Count > 10)

            {

                _tempDataSet.RemoveAt(0);

            }

            StateHasChanged();

        });


        await hubConnection.StartAsync();

    }

}
```

A.3.1.7 Файл компоненту SensorChartComponent.razor

```
@using ChartJs.Blazor.Charts

@using ChartJs.Blazor.ChartJS.Common.Properties

@using ChartJs.Blazor.ChartJS.Common.Enums

@using ChartJs.Blazor.ChartJS.Common.Axes

@using ChartJs.Blazor.ChartJS.Common.Axes.Ticks

@using ChartJs.Blazor.ChartJS.Common.Handlers

@using ChartJs.Blazor.ChartJS.Common.Time

@using ChartJs.Blazor.ChartJS.LineChart

@using ChartJs.Blazor.Util

@inject IJSRuntime JsRuntime

@using AirQualityControl.Shared.Entities

@using Microsoft.AspNetCore.SignalR.Client
```

```
@inject NavigationManager NavigationManager

@implements IDisposable

@using System.Net.Http

@inject HttpClient Http


<ChartJsLineChart @ref="_lineChartJs" Config="@_lineConfig"
Width="600" Height="300" />


@code {

    [Parameter]

    public int SensorId { get; set; }


    LineConfig _lineConfig;

    ChartJsLineChart _lineChartJs;

    LineDataset<TimeTuple<int>> _tempDataSet;


    private HubConnection hubConnection;


    public void Dispose()

    {

        _ = hubConnection.DisposeAsync();

    }


    protected override async Task OnInitializedAsync()

    {

        _lineConfig = new LineConfig

        {

            Options = new LineOptions
```

```
            {
                Animation = new Animation { Duration = 0 },
                Responsive = true,
                Title = new OptionsTitle
                {
                    Display = true,
                    Text = "Line Chart"
                },
                Legend = new Legend
                {
                    Position = Position.Right,
                    Labels = new LegendLabelConfiguration
                    {
                        UsePointStyle = true
                    }
                },
                Tooltips = new Tooltips
                {
                    Mode = InteractionMode.Nearest,
                    Intersect = false
                },
                Scales = new Scales
                {
                    xAxes = new List<CartesianAxis>
{
                        new TimeAxis
                        {
```

```
                                Distribution =
TimeDistribution.Linear,
                                Ticks = new TimeTicks
                                {
                                    Source = TickSource.Data
                                },
                                Time = new TimeOptions
                                {
                                    Unit =
TimeMeasurement.Millisecond,
                                    Round =
TimeMeasurement.Millisecond,
                                    TooltipFormat = "DD.MM.YYYY
HH:mm:ss:SSS",
                                    DisplayFormats =
TimeDisplayFormats.DE_CH
                                },
                                ScaleLabel = new ScaleLabel
                                {
                                    LabelString = "Time"
                                }
                            }
                        }
                    },
                    Hover = new LineOptionsHover
                    {
                        Intersect = true,
                        Mode = InteractionMode.Y
                    }
                }
```

```
        };


        _tempDataSet = new LineDataset<TimeTuple<int>>
        {
            BackgroundColor = ColorUtil.ColorString(0, 255, 0,
1.0),
            BorderColor = ColorUtil.ColorString(0, 0, 255, 1.0),
            Label = "CO2 in ppm",
            Fill = false,
            BorderWidth = 2,
            PointRadius = 3,
            PointBorderWidth = 1,
            SteppedLine = SteppedLine.False
        };


        _lineConfig.Data.Datasets.Add(_tempDataSet);


        var values = await
Http.GetFromJsonAsync<List<Value>>($"api/sensors/{SensorId}/valu
es?limit=10");
        values.OrderBy(p =>
p.DateTime.ToLocalTime()).ToList().ForEach(p =>
_tempDataSet.Add(new TimeTuple<int>(new
Moment(p.DateTime.ToLocalTime()), p.CO2)));


        hubConnection = new HubConnectionBuilder()
        .WithUrl(NavigationManager.ToAbsoluteUri("/valuehub"))
        .Build();


        hubConnection.On<Value>("ReceiveValue", (value) =>
        {
```

```
            if (value.SensorId != SensorId)
                return;

            _tempDataSet.Add(new TimeTuple<int>(new
Moment(value.DateTime.ToLocalTime()), value.CO2));
            if (_tempDataSet.Data.Count > 10)
            {
                _tempDataSet.RemoveAt(0);
            }
            StateHasChanged();
        });

        await hubConnection.StartAsync();
    }
}
```

A.3.1.8 Файл компоненту SensorComponent.razor

```
@using AirQualityControl.Shared.Entities
@inject NavigationManager NavigationManager
@using System.Net.Http
@inject HttpClient Http
@using AirQualityControl.Client.CombinedEntities
@using BlazorAnimate

@if (SensorWithCurrentValue == null ||
SensorWithCurrentValue.Sensor == null || Sector == null ||
SensorWithCurrentValue.Value == null)
{
    <SpinnerComponent/>
```

```
}

else

{

    <div class="card bg-light mb-3" style="max-width: 18rem;
min-height: 200px; min-width: 280px">

        <div class="card-header">

            Id: @SensorWithCurrentValue.Sensor.Id

            <span style="margin-left: 70px">STATUS:</span>

            @if ((DateTime.UtcNow -
SensorWithCurrentValue.Value.DateTime).TotalSeconds > 30)

            {

                <span class="text-warning"
style="">OFFLINE</span>

            }

            else

            {

                <span class="text-success"
style="">ONLINE</span>

            }

        </div>

        <div class="card-body">

            @if (SensorWithCurrentValue.VentilationFlapCommand
== null)

            {

                <Animate Animation="Animations.FadeIn"
Duration="TimeSpan.FromSeconds(0.5)">

                    <SpinnerComponent />

                </Animate>

            }

            else

            {
```

```
                <Animate Animation="Animations.FadeIn"
Duration="TimeSpan.FromSeconds(0.5)"
Delay="TimeSpan.FromMilliseconds(100)">

                    <h5 class="card-title">Ventilation Flap Open
Percentage: @FlapStatus</h5>

                    <div class="progress">

                        <div class="progress-bar"
role="progressbar" aria-valuenow="75" aria-valuemin="0" aria-
valuemax="100" style="width:@FlapStatus"></div>

                    </div>

                </Animate>

            }



            @if (SensorWithCurrentValue.Value == null)

            {

                <p class="card-text" style="margin-top: 16px">

                    <h6>Current CO2: No Data</h6>

                    <span>Last Sync Time: No Data</span>

                </p>

            }

            else

            {

                <p class="card-text" style="margin-top: 16px">

                    @if (SensorWithCurrentValue.Value.CO2 >
Sector.UpperThreshold)

                    {

                        <h6 class="text-danger">Current CO2:
@SensorWithCurrentValue.Value.CO2 ppm <span class="oi oi-
warning" aria-hidden="true"></span></h6>

                    }

                    else

                    {
```

```
                        <h6>Current CO2:
@SensorWithCurrentValue.Value.CO2 ppm</h6>

                    }

                    <span>Last Sync Time:
@SensorWithCurrentValue.Value.DateTime.ToLocalTime().ToShortTime
String()</span>

                </p>

            }

            <a href="/sensors/@SensorWithCurrentValue.Sensor.Id"
class="stretched-link"></a>

        </div>

    </div>

}




@code {

    [Parameter]

    public SensorWithCurrentValue SensorWithCurrentValue { get;
set; }



    [Parameter]

    public Sector Sector { get; set; }



    string FlapStatus =>
$"{SensorWithCurrentValue.VentilationFlapCommand.FlapLevel}%";

}
```

A.3.1.9 Файл компоненту SpinnerComponent.razor

```
<div class="d-flex justify-content-center">

    <div class="spinner-border" role="status">

        <span class="sr-only">Loading...</span>
```

```
    </div>
</div>
@code {


}
```

A.3.1.10 Файл сторінки ControlPanel.razor

```
@using AirQualityControl.Client.Components
@using AirQualityControl.Shared.Entities
@using BlazorAnimate
@using System.Net.Http
@inject HttpClient Http
@using Microsoft.AspNetCore.SignalR.Client
@inject NavigationManager NavigationManager
@implements IDisposable


<h3 style="text-align: center">Control Panel</h3>


@if (SectorWithCurrentFanSpeeds.Count == 0)
{
    <SpinnerComponent />
}
else
{
    <div style="display: flex; flex-direction: row; justify-
content: flex-start; flex-wrap: wrap;">
        @foreach (var sectorWithCurrentFanSpeed in
SectorWithCurrentFanSpeeds)
        {
```

```
            <div style="margin: auto">

                <Animate Animation="Animations.ZoomIn"
Duration="TimeSpan.FromSeconds(0.5)"
Delay="TimeSpan.FromMilliseconds(animationDelay+= 100)">

                    <SectorComponent
Sector="sectorWithCurrentFanSpeed.Sector"
CurrentFanSpeed="sectorWithCurrentFanSpeed.CurrentFanSpeed" />

                </Animate>

            </div>

        }

    </div>

}



@code {

    public List<SectorWithCurrentFanSpeed>
SectorWithCurrentFanSpeeds = new
List<SectorWithCurrentFanSpeed>();

    public int animationDelay = 0;



    private HubConnection hubConnection;



    protected override async Task OnInitializedAsync()

    {

        var sectors = await
Http.GetFromJsonAsync<List<Sector>>("api/sectors");

        sectors.ForEach(p => SectorWithCurrentFanSpeeds.Add(new
SectorWithCurrentFanSpeed { Sector = p }));



        hubConnection = new HubConnectionBuilder()


.WithUrl(NavigationManager.ToAbsoluteUri("/valuehub"))

            .Build();
```

```
        hubConnection.On<FanSpeedCommand>("ReceiveFanSpeed",
(fanSpeedCommand) =>

        {

            foreach (var sectorWithCurrentFanSpeed in
SectorWithCurrentFanSpeeds.Where(p => p.Sector.Id ==
fanSpeedCommand.SectorId))

            {

                sectorWithCurrentFanSpeed.CurrentFanSpeed =
fanSpeedCommand.Speed;

            }

            StateHasChanged();

        });



        await hubConnection.StartAsync();

    }



    public void Dispose()

    {

        _ = hubConnection.DisposeAsync();

    }



    public class SectorWithCurrentFanSpeed

    {

        public Sector Sector { get; set; }

        public int? CurrentFanSpeed { get; set; }

    }

}
```

A.3.1.11 Файл сторінки SectorDetailed.razor

```
@page "/sectors/{id:int}"

@using AirQualityControl.Client.Components

@using AirQualityControl.Shared.Entities

@using BlazorAnimate

@using System.Net.Http

@inject HttpClient Http

@implements IDisposable

@inject NavigationManager NavigationManager

@using Microsoft.AspNetCore.SignalR.Client

@using AirQualityControl.Client.CombinedEntities

@using ChartJs.Blazor.Charts

@using ChartJs.Blazor.ChartJS.Common.Properties

@using ChartJs.Blazor.ChartJS.Common.Enums

@using ChartJs.Blazor.ChartJS.Common.Axes

@using ChartJs.Blazor.ChartJS.Common.Axes.Ticks

@using ChartJs.Blazor.ChartJS.Common.Handlers

@using ChartJs.Blazor.ChartJS.Common.Time

@using ChartJs.Blazor.ChartJS.LineChart

@using ChartJs.Blazor.Util


<h3 style="text-align: center">Sector @Id "@Sector?.Name"</h3>

@if (isChartVisible)

{

    <button style="display: block" type="button" class="btn btn-
primary" @onclick="ChangeChartVisibility">Hide Charts</button>

    <div style="display: block"><SectorChartComponent
SectorId="Id" /></div>

    <div style="display: block"><SectorEnergyChartComponent
SectorId="Id" /></div>
```

```
}

else

{

    <button style="display: block" type="button" class="btn btn-
primary" @onclick="ChangeChartVisibility">Show Charts</button>

}

<h3 style="text-align: center">Sensors</h3>

<div style="display: flex; flex-direction: row; justify-content:
flex-start; flex-wrap: wrap">

    @foreach (var sensorWithCurrentValue in
SensorWithCurrentValues)

    {

        <div style="margin: 5px">

            <Animate Animation="Animations.ZoomIn"
Duration="TimeSpan.FromSeconds(0.5)"
Delay="TimeSpan.FromMilliseconds(animationDelay+= 100)">

                <SensorComponent
SensorWithCurrentValue="sensorWithCurrentValue" Sector="Sector"
/>

            </Animate>

        </div>

    }

</div>


@code {

    [Parameter]

    public int Id { get; set; }


    private bool isChartVisible;


    private HubConnection hubConnection;
```

```
    public List<SensorWithCurrentValue> SensorWithCurrentValues
= new List<SensorWithCurrentValue>();

    private int animationDelay = 0;



    private Sector Sector;



    private void ChangeChartVisibility()

    {

        isChartVisible = !isChartVisible;

    }



    protected override async Task OnInitializedAsync()

    {

        var sensors = await
Http.GetFromJsonAsync<List<Sensor>>($"api/sectors/{Id}/sensors")
;



        foreach (var sensor in sensors)

        {

            var value = await
Http.GetFromJsonAsync<List<Value>>($"api/sensors/{sensor.Id}/val
ues?limit=1");

            SensorWithCurrentValues.Add(new
SensorWithCurrentValue { Sensor = sensor, Value =
value.FirstOrDefault() });

        }



        Sector = await
Http.GetFromJsonAsync<Sector>($"api/sectors/{Id}");



        hubConnection = new HubConnectionBuilder()
```

```
.WithUrl(NavigationManager.ToAbsoluteUri("/valuehub"))
            .Build();


        hubConnection.On<Value>("ReceiveValue", (value) =>
        {
            foreach (var sensorWithCurrentValue in
SensorWithCurrentValues.Where(p => p.Sensor.Id ==
value.SensorId))
            {
                sensorWithCurrentValue.Value = value;
            }
            StateHasChanged();
        });



hubConnection.On<VentilationFlapCommand>("ReceiveFlapLevel",
(ventilationFlapCommand) =>
        {
            foreach (var sensorWithCurrentValue in
SensorWithCurrentValues.Where(p => p.Sensor.Id ==
ventilationFlapCommand.SensorId))
            {
                sensorWithCurrentValue.VentilationFlapCommand =
ventilationFlapCommand;
            }
            StateHasChanged();
        });


        await hubConnection.StartAsync();
    }
```

```
    public void Dispose()
    {
        _ = hubConnection.DisposeAsync();
    }
}
```

A.3.1.12 Файл сторінки SectorsEditor.razor

```
@page "/editor/sectors"
@using AirQualityControl.Client.Components;
@using AirQualityControl.Shared.Entities
@using System.Net.Http
@inject HttpClient Http

<table class="table">
    <thead>
        <tr>
            <th scope="col">Id</th>
            <th scope="col">Name</th>
            <th scope="col">Lower Threshold</th>
            <th scope="col">Upper Threshold</th>
            <th scope="col">Actions</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <th scope="row"><input type="number" min="1"
@bind="@sector.Id"/></th>
            <td><input @bind="@sector.Name" /></td>
```

```
            <td><input type="number"
@bind="@sector.LowerThreshold"/></td>

            <td><input type="number"
@bind="@sector.UpperThreshold"/></td>

            <td>

                @if (Sectors.Select(p =>
p.Id).Contains(sector.Id))

                {

                    <button type="button" class="btn btn-
primary" disabled @onclick="AddSector">Add</button>

                }

                else

                {

                    <button type="button" class="btn btn-
primary" @onclick="AddSector">Add</button>

                }

            </td>

        </tr>

        @foreach (var sector in Sectors)

        {

            <EditSectorComponent Sector="sector"
SectorsEditor="this" />

        }

    </tbody>

</table>



@code {

    public List<Sector> Sectors { get; set; } = new
List<Sector>();

    private Sector sector = new Sector();
```

```
    protected override async Task OnInitializedAsync()

    {

        Sectors = await
Http.GetFromJsonAsync<List<Sector>>("api/sectors");



        sector = new Sector { Id = Sectors.LastOrDefault().Id +
1 };

    }



    private async Task  AddSector()

    {

        var httpResponseMessage = await
Http.PostAsJsonAsync("api/sectors", sector);



        if(httpResponseMessage.StatusCode ==
System.Net.HttpStatusCode.OK)

        {

            Sectors.Add(sector);

        }



        sector = new Sector { Id = Sectors.LastOrDefault().Id +
1 };

    }



    public void DeleteSector(Sector sector)

    {

        Sectors.Remove(sector);

        StateHasChanged();

    }

}
```

А.3.1.13 Файл сторінки SensorDetailed.razor

```
@page "/sensors/{id:int}"
@using AirQualityControl.Client.Components;

<h3>Sensor @Id</h3>

<SensorChartComponent SensorId="Id"/>

@code {
    [Parameter]
    public int Id { get; set; }
}
```

А.3.1.14 Файл сторінки SensorsEditor.razor

```
@page "/editor/sensors"
@using AirQualityControl.Client.Components;
@using AirQualityControl.Shared.Entities
@using System.Net.Http
@inject HttpClient Http

<table class="table">
    <thead>
        <tr>
            <th scope="col">Id</th>
            <th scope="col">SectorId</th>
            <th scope="col">Actions</th>
        </tr>
    </thead>
```

```
    <tbody>
        @if (Sectors != null)
        {
            <tr>
                <th scope="row"><input type="number" min="1"
@bind="@sensor.Id" /></th>
                <td style="width:800px">
                    <select class="custom-select"
@bind="@sensor.SectorId">
                        @foreach (var id in Sectors.Select(p =>
p.Id))
                        {
                            <option value="@id">@id</option>
                        }
                    </select>
                </td>
                <td>
                    <button type="button" class="btn btn-
primary" @onclick="AddSensor">Add</button>
                </td>
            </tr>
        }
        @if (Sensors != null)
        {
            @foreach (var sensor in Sensors)
            {
                <EditSensorComponent Sensor="sensor"
Sectors="Sectors" SensorsEditor="this" />
            }
        }
```

```
    </tbody>

</table>


@code {

    public List<Sensor> Sensors { get; set; } = new
List<Sensor>();

    public List<Sector> Sectors { get; set; } = new
List<Sector>();



    private Sensor sensor = new Sensor();



    protected override async Task OnInitializedAsync()

    {

        Sensors = await
Http.GetFromJsonAsync<List<Sensor>>("api/sensors");

        Sectors = await
Http.GetFromJsonAsync<List<Sector>>("api/sectors");



        sensor = new Sensor { Id = Sensors.LastOrDefault().Id +
1, SectorId = Sectors.FirstOrDefault().Id };

    }



    private async Task AddSensor()

    {

        var httpResponseMessage = await
Http.PostAsJsonAsync("api/sensors", sensor);



        if (httpResponseMessage.StatusCode ==
System.Net.HttpStatusCode.OK)

        {

            Sensors.Add(sensor);
```

```
        }


        sensor = new Sensor { Id = Sensors.OrderByDescending(p
=> p.Id).FirstOrDefault().Id + 1, SectorId =
Sectors.FirstOrDefault().Id };

    }


    public void DeleteSensor(Sensor sensor)

    {

        Sensors.Remove(sensor);

        StateHasChanged();

    }

}
```

ДОДАТОК Б. СПИСОК ПУБЛІКАЦІЙ

Б.1 XIII Науково-практична конференція студентів, аспірантів та молодих вчених «Погляд у майбутнє приладобудування»-2020

УДК 681.518.3

Д.М. Семенко, студент гр. ПА-91мп, к.т.н., доц. Стаценко О.В.
КПІ ім. Ігоря Сікорського

ЕНЕРГОЗБЕРЕЖЕННЯ ЗАСОБАМИ АВТОМАТИЗОВАНИХ ВЕНТИЛЯЦІЙНИХ СИСТЕМ

Анотація. В статті розглянуті можливості енергозбереження при використанні вентиляційних систем. Наведені переваги використання під час керування вентиляцією принципу «за потребою», проаналізовані існуючі системи вентиляції, проведена оцінка можливості зниження споживаної енергії при використанні регульованого електроприводу вентилятора.

Ключові слова : вентиляція, енергозбереження, автоматизація.

ВСТУП

Найбільш екологічним та чистим джерелом енергії є енергозбереження. Це обумовлено зменшенням споживання енергії, яка виробляється різними видами електростанцій. Автоматизація інженерних систем будівель - це важливий інструмент в боротьбі з нераціональним використанням енергоресурсів і забрудненням навколишнього середовища. Також це допомагає у створенні комфортного мікроклімату всередині приміщень. Будівля, оснащена новими сучасними системами автоматизації, це будівля, що представляє собою не застиглу архітектуру, а структуру зі складними системами життєзабезпечення.

Вентиляція виконує роль створення обміну повітря в приміщенні для видалення надлишків шкідливих речовин, вологи, теплоти та забезпечення допустимих умов повітряного середовища. У житлових та офісних приміщеннях вентиляційні системи забезпечують повітрообмін в приміщенні, видаляючи з нього повітря з підвищеною концентрацією вуглекислого газу і різними запахами і наповнюючи свіжим повітрям, які пройшли певну підготовку (очищення, нагрівання).

Основними типами сучасних систем вентиляції є [1]:

1. Природна вентиляція. Найбільш проста вентиляційна система, яка не потребує додаткового обладнання, що забезпечує повітрообмін за допомогою різниці температур і тиску повітря всередині і поза приміщенням, швидкість вітру і т.д. Вентиляція повітря здійснюється за допомогою спеціальних вентиляційних отворів, а також не герметичних віконних прорізів.
2. Штучна система вентиляції передбачає використання спеціального обладнання (клапанів, фільтрів, обігрівачів повітря, вентиляторів), за допомогою якого забезпечується ефективна вентиляція в будь-який час року, незалежно від погодних умов. До видів штучної вентиляції відносять:
   a. Припливно-витяжну систему, яка забезпечує виведення використаного повітря і подачу замість свіжого. Крім загальнообмінної вентиляції, що забезпечує свіжим повітрям всієї будівлі, може бути місцева вентиляція, що забезпечує обмін повітря в окремих приміщеннях.
   b. Моноблочну систему, яка об'єднує всі компоненти в єдиному шумоізольованому корпусі, подаючи по воздуховодам вже оброблене повітря.
   c. Набірну систему, що складається з окремих компонентів, має складним проектуванням і значні габарити.

При проектуванні систем вентиляції користуються нормативними документами, основними з яких є Державні будівельні норми України [2,3]. Згідно з цими нормами продуктивність системи вентиляції має складати 20-60 м^3 на годину на одну людину в залежності від типу приміщення. Одночасно з цим якість повітря регламентується державними стандартами, згідно з якими допускається використання вентиляції за потребою. Це є надзвичайно актуальним при змінній кількості людей в приміщенні, наприклад в кінотеатрах, концертних залах, спортзалах та інше.

Такий підхід до організації систем вентиляції відповідає найжорсткішим вимогам енергоефективності та в порівнянні з системою постійної витрати повітря дозволяє заощадити до 80% енергоспоживання вентиляторів і до 40% енергії на охолодження і обігрів [4].

ОСНОВНА ЧАСТИНА

На сьогодні різні компанії пропонують комплексні рішення побудови систем вентиляції. Наприклад компанія Swegon пропонує для рішення для організації вентиляції в навчальних кабінетах та конференц-залах, котре передбачає організацію вентиляції за потреби з використанням такого обладнання: регульовані приводні заслінки з датчиками якості повітря, дифузори припливного та відпрацьованого повітря, датчики присутності, клапани радіаторів опалення. Використання такого рішення дозволяє зменшити теплові витрати, та енерговитрати, за умови наявності керованої системи централізованої вентиляції. За відсутності такої системи, або при використанні місцевої вентиляції, такий підхід не забезпечує енергоефективну роботу.

Згідно з [5] кожен вентилятор характеризується залежностями напору та потужності від подачі повітря при постійній частоті обертання. Приклад цих характеристик наведений на рисунку 1. В загальному випадку коефіцієнт корисної дії має чітко виражений екстремум і в номінальному режимі роботи робоча точка вентилятора має відповідати максимуму ККД. При перекриванні вентиляційного каналу, наприклад при перекритті заслінок, змінюються характеристики вентиляційної системи і робоча точка зміщується, що призводить до зменшення ККД. Споживана потужність при цьому також знижується, але не пропорційно зменшенню подачі повітря. Для підтримання ККД на постійному рівні необхідно змінювати частоту обертання вентилятора. При цьому подача повітря змінюється пропорційно частоті обертання, а потужність змінюється пропорційно кубу від частоти. Саме тому використання частотно-регульованого електроприводу вентилятора забезпечує найвищий рівень енергозбереження.

Розглянемо який рівень електроенергії буде споживатися двома системами: в одній використовуватиметься регулювання подачі повітря шляхом перекриття заслінок, а в іншій використовуватиметься регулювання шляхом зміни частоти обертання.

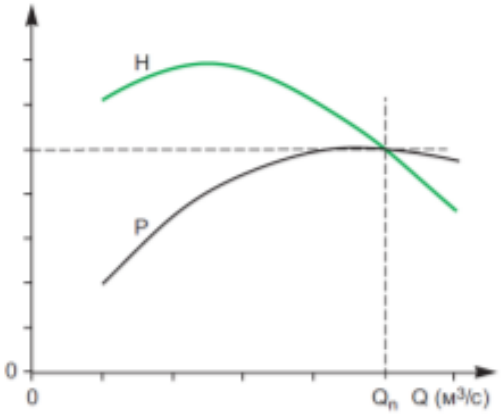

Рисунок 1. Характеристики напору H та потужності P від подачі повітря Q

Вважатимемо, що в деякому приміщенні є потреба у подачі повітря за добу складає 100% протягом 2 годин, 90% протягом 8 годин та 50% протягом решти 14 годин. Залежності зміни потужності для деякого вентилятора при першому та другому підходах на рисунку 2 (суцільною лінією – перший підхід, пунктиром - другий).

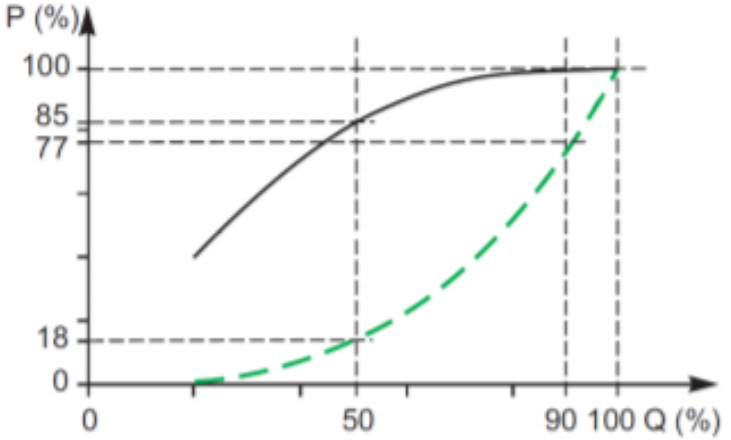

Рисунок 2. Залежності потужності споживаної вентилятором при різних способах регулювання подачі повітря

Для оцінки середньодобової відносної потужності споживаної з мережі енергії необхідно враховувати додатково ККД електричного двигуна вентилятора та для другого випадку регулювання ККД перетворювача частоти. ККД асинхронного двигуна залежить від його номінальної потужності, режиму роботи та навантаження. Для двигуна потужністю 90 кВт при живленні від мережі ККД складає від 89 % до 93 % по мірі збільшення навантаження від 25 % до 100 % [6]. При живленні від перетворювача частоти ККД двигуна залежить від алгоритму керування і є більш стабільним, тобто можна вважати для того

самого двигуна рівним 93 %. ККД перетворювача частоти може складати від 90 % до 95 % в залежності від конкретного пристрою (для розрахунку приймемо 90 %). Розрахунок проведемо згідно з формулою (1) для першого випадку, і формулою (2) для другого:

$$P_{сер1} = \frac{\frac{P'_{100\%} \cdot 2}{\eta_{дв100\%}} + \frac{P'_{90\%} \cdot 8}{\eta_{дв90\%}} + \frac{P'_{50\%} \cdot 14}{\eta_{дв50\%}}}{24} = \frac{\frac{100\% \cdot 2}{0.93} + \frac{98\% \cdot 8}{0.93} + \frac{85\% \cdot 14}{0.9}}{24} = 91.3\% \quad (1)$$

$$P_{сер2} = \frac{\frac{P''_{100\%} \cdot 2}{\eta_{дв} \cdot \eta_{пер}} + \frac{P''_{90\%} \cdot 8}{\eta_{дв} \cdot \eta_{пер}} + \frac{P''_{50\%} \cdot 14}{\eta_{дв} \cdot \eta_{пер}}}{24} = \frac{\frac{100\% \cdot 2}{0.93 \cdot 0.9} + \frac{77\% \cdot 8}{0.93 \cdot 0.9} + \frac{18\% \cdot 14}{0.93 \cdot 0.9}}{24} = 51.3\% \quad (2)$$

Як видно з отриманих результатів зниження середньодобової відносної потужності при використанні другого підходу до регулювання подачі повітря складає 44 % порівняно з першим підходом. Слід зазначити, що в ряді випадків необхідним є ще більше зниження подачі повітря, що призведе до ще більшої економії електроенергії.

ВИСНОВКИ

В статті розглянуті основні можливості забезпечення енергозбереження засобами електроприводу вентиляційних систем. Показано, що використання регульованого електроприводу місцевої вентиляції дозволяє суттєво знизити споживану потужність при регулюванні подачі повітря за потребою.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Системи вентиляції. Терміни та визначення: ДСТУ 2388-94. — [Введ. в дію 17.04.1995]. — К. : Держстандарт України, 1994. — 49 с.

[2] Опалення, вентиляція та кондиціонування: ДБН В.2.5-67:2013. — [Введ. в дію 25.01.2013]. — К. : Державні будівельні норми України, 2013, — 24 с.

[3] Здания жилые и общественные. Параметры микроклимата в помещениях: ГОСТ 30494-2011 — [Введ. в дію 01.01.2013]. — Межгосударственный стандарт, 1994. — 191 с.

[4] Ягьяева Л. Т., Ахметханов А. А. Автоматизированная система управления приточно-вытяжной вентиляции / Л. Т. Ягьяева, А. А. Ахметханов // Вестник Казанского технологического университет, 2013. – № 22, том 16, с.264-266.

[5] Автоматизированный электропривод типовых производственных механизмов и технологических комплексов / М.П. Белов, В.А. Новиков, Л.Н. Рассудов – М.: Изд.центр «Академия», 2007 – 576 с.

[6] Асинхронные двигатели серии 4А: Справочник / А.Э. Кравчик, М.М. Шлаф, В.И. Афонин, Е.А. Соболенская. – М.: Энергоиздат, 1982. – 504 с.

Наук. керівник – к.т.н., доц. Стаценко О.В.

Б.2 XVI Науково-практична конференція студентів, аспірантів та молодих вчених «Ефективність та автоматизація інженерних рішень у приладобудуванні» -2020.

УДК 681.518.3

Д.М. Семенко, студент гр. ПА-91мп, к.т.н., доц. Стаценко О.В.
КПІ ім. Ігоря Сікорського

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ КОНТРОЛЮ ЯКОСТІ ПОВІТРЯ В СИСТЕМАХ ВЕНТИЛЯЦІЇ ІНТЕЛЕКТУАЛЬНИХ БУДІВЕЛЬ

Анотація. У статті розглянута структура системи вентиляції інтелектуальних будівель та визначені вимоги до керування такими системами. Проаналізовані існуючі підходи до передачі даних в таких системах, та визначені найбільш підходящі інструменти до розробки програмного забезпечення цих систем.

Ключові слова: вентиляція, система контролю, передача та відображення даних.

ВСТУП

На сьогодні існує декілька різних визначень поняття «інтелектуальна будівля» [1]. Згідно з одним з них, інтелектуальною будівлею є така будівля, в якій використовуються спеціальні технології для створення більш безпечного, комфортного та продуктивного середовища для мешканців, та збільшується ефективність її експлуатації для власників. Більшість сучасних громадських та житлових будівель планується з метою зменшення витрат за рахунок зменшення впровадження енергоефективних технологій. Одним з основних способів зменшення витрат енергії є використання вентиляційних систем, керування якими здійснюється в залежності від потреби [2]. До складу вентиляційних систем інтелектуальних будівель входять:

- повітроводи з встановленими вентиляторами, швидкість обертання яких, а, відповідно, і продуктивність, регулюється за допомогою автоматизованих електроприводів;
- комплекти обладнання в окремих приміщеннях, які складаються з керованих заслінок, датчиків вмісту вуглекислого повітря та спеціалізованих контролерів;
- системи моніторингу параметрів роботи всіх інженерних мереж будівлі.

Для керування такою системою вентиляції та контролюванням її роботи необхідним є створення системи збору даних про параметри якості повітря в окремих приміщеннях будівлі та параметри стану кожного з елементів системи вентиляції. Вирішення такої задачі передбачає визначення структури апаратної складової частини системи та розробку програмної складової. Апаратна частина може бути побудована на базі спеціалізованих контролерів, встановлених в окремих приміщеннях та пристроїв керування електроприводами, об’єднаними за допомогою локальної мережі. Для розробки програмної складової необхідним є вибір механізму обміну даними, а також визначення найбільш ефективного інструментарію для безпосередньої розробки програмного додатку.

ПОСТАНОВКА ЗАДАЧІ

Вибір та обґрунтування способу передачі даних та розробка програмного клієнтського додатку для відображення параметрів роботи автоматизованої системи керування вентиляцією інтелектуальних будівель.

ОСНОВНИЙ МАТЕРІАЛ

Обмін даними з використанням локальних або глобальних мереж наразі здійснюється з використанням на транспортному рівні протоколу TCP (Transmission Control Protocol – протокол керування передачею), а на прикладному рівні протоколу HTTP (HyperText Transfer Protocol – протокол передачі гіпертексту). Протокол HTTP був створений для зв'язку між веб-браузерами і веб-серверами, але в принципі він може використовуватися і для інших цілей [3].

Широке використання даного протоколу призвело до формування різних підходів його використання, найбільш розповсюдженими серед яких є: HTTP Polling та HTTP Long Polling.

HTTP Polling є дещо застарілим підходом, але він все ж таки ще використовується. Суть передачі даних з використанням такого підходу полягає у послідовному формуванні запиті даних через деякий проміжок часу. У цього методу є багато мінусів. Самі явні з них:

- на частину запитів приходять «порожні» відповіді, що призводить до непродуктивного використання мережі;
- часова затримка між відправленнями запитів призводить до затримки отримання відповідей, що може негативно вплинути на роботу системи.

Покращений варіант попереднього методу HTTP Long Polling полягає в тому, що клієнт відправляє запит на сервер, сервер тримає відкритим з'єднання поки не прийдуть якісь дані або клієнт не відключиться самостійно. Як тільки дані прийшли – відправляється відповідь і з'єднання закривається. Переваги даного методу порівняно з HTTP Polling:

- зменшується кількість запитів, оскільки відсутні «порожні» відповіді;
- підвищується часова точність подій, оскільки передача даних ініціюється клієнтом;
- сервер зберігає події тільки на час перепідключення.

Але для організації обміну даними в розглядуваному випадку існує краще рішення, яке полягає у використанні протоколу прикладного рівня WebSocket. Це технологія, що дозволяє відкрити постійне двунаправлене мережеве з'єднання між браузером користувача та сервером. За допомогою його API (Application Programming Interface) можна відправити повідомлення на сервер і отримати відповідь без виконання http-запиту, причому цей процес буде подієво-керованим [4]. Переваги в порівнянні з HTTP Long Polling:

- встановлюється лише одне з'єднання;
- забезпечується гранично висока часова точність подій.

При виборі такого протоколу розробку серверного програмного забезпечення доцільно здійснювати з використанням платформи .NET та бібліотеки SignalR. Ця бібліотека надає простий API для створення віддалених викликів процедур від сервера до клієнта, які викликають функції JavaScript у браузерах клієнтів (та інших клієнтських платформах) із коду .NET на стороні сервера. SignalR також включає API для управління з'єднаннями та групування з'єднань. SignalR використовує протокол WebSocket там, де він доступний, і повертається до старих протоколів, де це необхідно [5].

Для розробки клієнтського програмного забезпечення разом із обраною бібліотекою доцільним є використання фреймворку Blazor. Він представляє UI-фреймворк для створення інтерактивних додатків, які можуть працювати як на стороні сервера, так і на стороні клієнта, на платформі .NET.

Blazor надає розробникам наступні переваги:

- написання коду веб-додатків за допомогою C # замість JavaScript;
- використання можливостей екосистеми .NET, зокрема, бібліотек .NET при створенні додатків;
- використання Visual Studio в якості інструменту для розробки, який має вбудовані шаблони для спрощення створення додатків.

З урахуванням проведеного аналізу структура системи має такий вигляд:

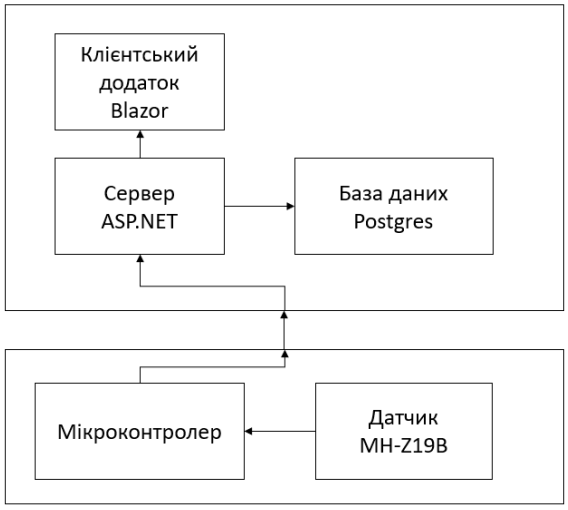

Рисунок 1 – Структурна схема системи контроля якості повітря

На рис.1 структурно показаний зв’язок програмної складової з вимірювальним модулем в одному приміщенні будівлі, при цьому передбачається, що вимірювальні модулі в інших приміщеннях мають таку саму структуру, а під час передачі даних використовують інші ідентифікатори.

Програмний продукт складається з таких компонентів: бази даних Postgres, серверної частини ASP.NET та клієнтського додатку. Дані від датчиків та параметри роботи системи вентиляції передаються з використанням мікроконтролерів до серверу. Реляційна база даних

використовується для зберігання цих даних, а також списку датчиків. Сервер обробляє ці дані та вираховує потужність вентилятора, яку треба встановити в тому чи іншому приміщені будівлі.

Розроблений клієнтський додаток має декілька складових частин, які мають вигляд наведений на рис.2,3.

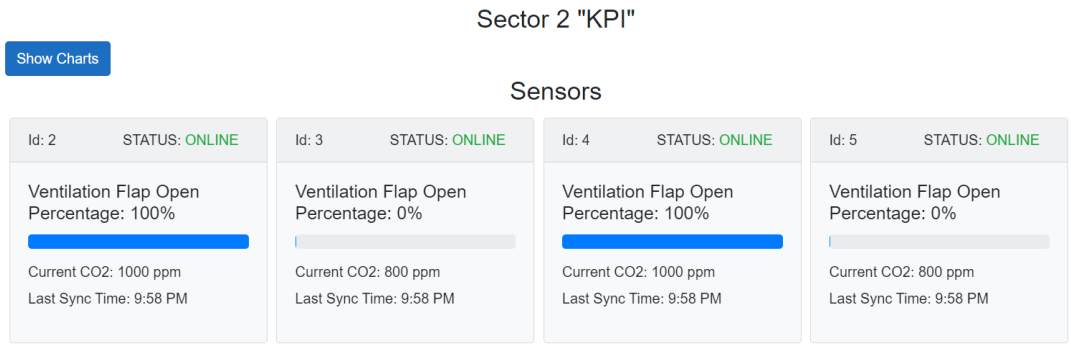

Рисунок 2 – Список датчиків в приміщеннях одного сектору з виводом поточних даних

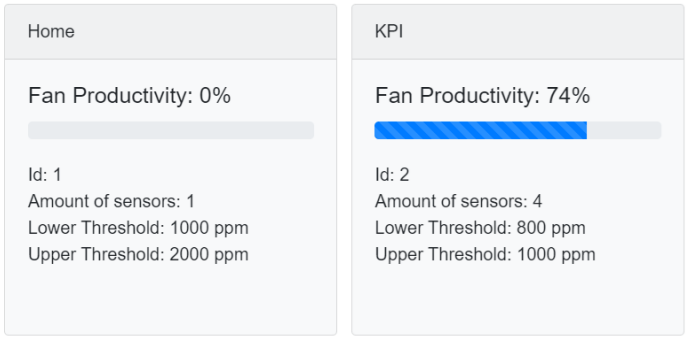

Рисунок 3 – Список секторів та відображення параметрів роботи системи вентиляції

ВИСНОВОК

В роботі проведений аналіз протоколам передачі даних, що можуть бути використані при побудові системи керування та діагностики роботи вентиляції інтелектуальних будівель, а також розроблений клієнтський програмний додаток на основі фреймворку Blazor.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Osama Omar. Intelligent building, definitions, factors and evaluation criteria of selection // Alexandria Engineering Journal, (2018) 57, pp.2903–2910.

[2] Семенко Д.М. Енергозбереження засобами автоматизованих вентиляційних систем / Д.М. Семенко, О.В. Стаценко // Збірник праць XIII Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених “ПОГЛЯД У МАЙБУТНЄ ПРИЛАДОБУДУВАННЯ” – К.: ПБФ, КПІ ім. Ігоря Сікорського, Центр учбової літератури. – 2020. – С.389-392.

[3] Приложения реального времени и Polling ,Long Polling , WebSockets, Server-Sent Events SSE.// Режим доступу: https://intellect.icu/prilozheniya-realnogo-vremeni-i-polling-long-polling-websockets-server-sent-events-sse-7016/

[4] WebSockets. MDN Web Docs. // Режим доступу: https://developer.mozilla.org/ru/docs/WebSockets/
[5] Patrick Fletcher. Introduction to SignalR. // Режим доступу: https://docs.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr

Наук. керівник – к.т.н., доц. Стаценко О.В.